NAVAL POSTGRADUATE SCHOOL

MONTEREY, CALIFORNIA

THESIS

DISTRIBUTED PASSWORD CRACKING

by

John R. Crumpacker

December 2009

Thesis Advisor: George Dinolt
Second Reader: Chris Eagle

Approved for public release; distribution is unlimited

REPORT DOCUMENTATION PAGE			Form Approved OMB No. 0704-0188
Public reporting burden for this collection of information is estimated to average 1 hour per response, including the time for reviewing instruction, searching existing data sources, gathering and maintaining the data needed, and completing and reviewing the collection of information. Send comments regarding this burden estimate or any other aspect of this collection of information, including suggestions for reducing this burden, to Washington headquarters Services, Directorate for Information Operations and Reports, 1215 Jefferson Davis Highway, Suite 1204, Arlington, VA 22202-4302, and to the Office of Management and Budget, Paperwork Reduction Project (0704-0188) Washington DC 20503.			
1. AGENCY USE ONLY ***(Leave blank)***	**2. REPORT DATE** December 2009	**3. REPORT TYPE AND DATES COVERED** Master's Thesis	
4. TITLE AND SUBTITLE: Distributed Password Cracking			**5. FUNDING NUMBERS**
6. AUTHOR(S) John R. Crumpacker			
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) Naval Postgraduate School Monterey, CA 93943-5000			**8. PERFORMING ORGANIZATION REPORT NUMBER**
9. SPONSORING /MONITORING AGENCY NAME(S) AND ADDRESS(ES) N/A			**10. SPONSORING/MONITORING AGENCY REPORT NUMBER**
11. SUPPLEMENTARY NOTES The views expressed in this thesis are those of the author and do not reflect the official policy or position of the Department of Defense or the U.S. Government.			
12a. DISTRIBUTION / AVAILABILITY STATEMENT Approved for public release; distribution is unlimited			**12b. DISTRIBUTION CODE**

13. ABSTRACT (maximum 200 words)

Password cracking requires significant processing power which in today's world is located at a workstation or home in the form of a desktop computer. Berkeley Open Infrastructure for Network Computing (BOINC) is the conduit to this significant source of processing power and John the Ripper is the key.

BOINC is a distributed data processing system that incorporates client-server relationships to generically process data. The BOINC structure supports any system that requires large amounts of data to be processed without changing significant portions of the structure.

John the Ripper is a password cracking program that takes a password file and attempts to determine the password by a guess and check method.

The merger of these two programs enables companies and diverse groups to verify the strength of their password security policy. This thesis goes into detail on the inner workings of BOINC, John the Ripper, and the merger of the two programs. It also details the work required to test the system to its full capability.

14. SUBJECT TERMS Distributed Password Cracking, Berkeley Open Infrastructure for Network Computing (BOINC), and John the Ripper			**15. NUMBER OF PAGES** 269
			16. PRICE CODE
17. SECURITY CLASSIFICATION OF REPORT Unclassified	**18. SECURITY CLASSIFICATION OF THIS PAGE** Unclassified	**19. SECURITY CLASSIFICATION OF ABSTRACT** Unclassified	**20. LIMITATION OF ABSTRACT** UU

NSN 7540-01-280-5500 Standard Form 298 (Rev. 2-89)
Prescribed by ANSI Std. 239-18

THIS PAGE INTENTIONALLY LEFT BLANK

Approved for public release; distribution is unlimited

DISTRIBUTED PASSWORD CRACKING

John R. Crumpacker
Lieutenant, United States Navy
B.S., United States Naval Academy, 2004

Submitted in partial fulfillment of the
requirements for the degree of

MASTER OF SCIENCE IN COMPUTER SCIENCE

from the

NAVAL POSTGRADUATE SCHOOL
December 2009

Author: John R. Crumpacker

Approved by: George Dinolt
Thesis Advisor

Chris Eagle
Second Reader

Dr. Peter J. Denning
Chairman, Department of Computer Science

THIS PAGE INTENTIONALLY LEFT BLANK

ABSTRACT

Password cracking requires significant processing power, which in today's world is located at a workstation or home in the form of a desktop computer. Berkeley Open Infrastructure for Network Computing (BOINC) is the conduit to this significant source of processing power and John the Ripper is the key.

BOINC is a distributed data processing system that incorporates client-server relationships to generically process data. The BOINC structure supports any system that requires large amounts of data to be processed without changing significant portions of the structure.

John the Ripper is a password cracking program that takes a password file and attempts to determine the password by a guess and check method.

The merger of these two programs enables companies and diverse groups to verify the strength of their password security policy. This thesis goes into detail on the inner workings of BOINC, John the Ripper, and the merger of the two programs. It also details the work required to test the system to its full capability.

THIS PAGE INTENTIONALLY LEFT BLANK

TABLE OF CONTENTS

LIST OF FIGURES

THIS PAGE INTENTIONALLY LEFT BLANK

ACKNOWLEDGMENTS

I am heartily thankful to my advisor, Dr. George Dinolt, whose patience, guidance, and support from the initial to the final stages of this endeavor enabled me to successfully complete this thesis.

I am grateful to Dr. Chris Eagle for his support in completing this thesis.

Finally, I offer my regards to all of those who supported me in any respect during the completion of the project.

THIS PAGE INTENTIONALLY LEFT BLANK

I. INTRODUCTION

A. THE PASSWORD PROBLEM

One of the easiest ways to breach a computer system is through normal user validation interfaces. Except for hardware security measures such as finger printer readers, smart cards, or tokens, the main way to validate users on a computer system is via password authentication. Since users cannot remember 128bit passwords or codes for authentication, they normally use a password that is easy for them to remember but is still a "strong password" (McDowell, 2009). The problem is most users do not follow good password selection processes, and the only way to check the strength of a password is before it is set or to attempt to break it afterwards, with the use of programs like John the Ripper (JtR). Organizational password policies are better now, requiring special characters, numbers, and capital letters. The password setting programs can check to ensure the presence of these characters, but most do not check the context or simple character manipulations.

The hardest part of this process is getting users to understand the importance of a good password. Users need to understand how passwords are managed and how hackers reconstruct passwords from the stored data (Garfinkel, n.d.).

Passwords of a system are stored as a hash on that system, or in a central location in some networks. A hash is the output of function that is one-way, $f(x) = y$. Therefore, easy to compute. However, since it is a one-way function, there is no tractable function $f^{-1}(y) = x$ to return the original password (Lim, 2004).

Even before the emergence of distributed computing, all passwords could be broken. Passwords with a simple structure, like dictionary words, were broken even faster than other passwords. Trying to crack user passwords is one way to ensure password strength. Since this is a computationally intense process, developing a distributed approach using idle computer processing power to determine the strength of network keys would be a good approach for many organizations. This approach will also help to compete with hackers or foreign governments using distributed computing power

to break passwords. By using the organization's excess processing power, the organization can adequately assess the strength of the organization's password policy as well as adherence to the policy.

Hackers use system and application flaws to gain access to a computer system. Then they download the password file to an off-line computer, where they attempt to retrieve the passwords by using a brute force "guess and check" method. Weak passwords, those without special characters such as numbers and/or capital letters, or passwords with simple manipulations (man becomes m@N, even becomes 3v3n, etc.), can be cracked by a single computer in a maximum of one month. However, with several computers working on a single weak password, that time can be divided by the number of computers used.

Consider a strong password that has special characters, numbers, lower case letters, and upper case letters. This will allow for 95 distinct possible characters. A password that is *n* characters in length will then have 95^n possible password combinations. With an eight-character password there are 6.63×10^{15} possible passwords. If you can process 100,000 passwords in a second then it would take about 2,000 years to check every possibility. On average the password can be found in half that time, but this is still far outside the scope of a normal password changing policy. Now consider if 10,000 computers checking 100,000 passwords a second work on that password. The time to check all possible passwords is about 2.6 months. Even with a strong password this is inside the time limits to change passwords for a normal password policy. With more computers working on the password and with the increased speed of computers today it is not unconceivable to have the ability to crack even passwords built with these rules in hours or days.

The focus of this research was to analyze and improve the process of password cracking by using the Berkeley Open Infrastructure for Network Computing (BOINC), a distributed computing system. Specifically, this research explores how to implement this distributed approach and compares it with other approaches.

As part of this work, the current implementation of John the Ripper, a password cracking program, was merged with BOINC.

B. DISTRIBUTED COMPUTING

There are several different groups developing generic algorithms and protocols for distributed computation. Most of them are designed to use simple message passing between a controlling computer and a client computer. Distributed.net (2003), Simple Object Access Protocol (SOAP, 2007), and Message Passing Interface (MPI) (Pippin et al., 2003) are just a few of the others besides BOINC. However, Distributed.net does not allow independent application development. SOAP (2007) does not support the data collection and analysis that BOINC does. Nor does SOAP (2007) offer the background execution operations offered by BOINC. MPI is designed for multiprocessor computers like clusters or mainframes that are dedicated to the distributed application. Since this project will not be supported by dedicated computers, MPI is not a viable solution for the environment in which this application will be running.

SETI@home was the first real distributed application that used standard home computers spare processing power to process an enormous amount of information. Because this approach was so popular the developer of SETI@home started to develop a general distributed computing system. From this initiative came BOINC a distributed system that is not proprietary and relatively easy to use. We decided to use BOINC for the distributed backbone of this application.

Since the conception of this project systems have been developed that enable organizations to have access to significant processing power and distributed password cracking capabilities. First, the FRED SC, developed by Digital Intelligence, is the first super computer available to the general public. This system offers 4.769 Tflops[1] of processing power that was specifically designed to aid in forensic computations for password cracking (Digital Intelligence, n.d.). Secondly, Elcomsoft Proactive Software developed the Elcomsoft Distributed Password Recovery software package, which enables a company to link up to 10,000 computers together to recover data locked by passwords (Elcomsoft Co., n.d.). These two systems are both proprietary methods for cracking passwords, but Distributed John the Ripper has some advantages over these products. First,

[1] Tflops (teraflops): A million million floating-point operations per second.

all the software and tools required to run Distributed JtR are open source. Secondly, it is scalable above the 10,000 computers. Therefore, Distributed JtR is a viable option for groups or individuals to provide the same capabilities.

C. PASSWORD CRACKING PROGRAMS

Most password cracking programs operate under the same principles. First, the program guesses a password, then checks to see if that password is correct. Other password cracking programs use rainbow tables which are large tables of pre-generated hashes consisting of nearly every possible password. Using rainbow tables removes a significant portion of time required to calculate the hash in a normal password cracking program. However, should a salt[2] be added to a password when it is stored, the table of hashed passwords is no longer valid and would have to be recalculated for the salt that was used for that password. Since each password can use a different salt, it becomes infeasible to store rainbow tables for each salt (Wikipedia-SALT, 2009).

To guess a password most password crackers compare against a dictionary list or login information to check for simple passwords. However, other password crackers, like John the Ripper, run each word in the dictionary through a list of rules that manipulates the word checking for simple substitutions (like $'s for S's). To check the password the program hashes the guess using the password hashing function. If the two hashes match the password is found. The real difference between password cracking programs is the way password guesses are generated and how the key-space is traversed.

D. CURRENT DISTRIBUTED PASSWORD CRACKERS

Other distributed password crackers have been implemented, some specifically for John the Ripper. The first parallelization of John the Ripper (JtR) was developed by Ryan Lim of the University of Nebraska in Lincoln; he successfully implemented JtR using MPI[3] for only the brute force mode of the JtR. Lim (2004) showed a linear decrease in time with the addition of multiple processors. The second parallelization of John the Ripper was again implemented with MPI (John-MPI), but this time only focused

[2] A salt is number that is added to the front of a password to cause the output of the hash function to be different from the hash without the salt.

[3] MPI - Message Passing Interface.

on finding weak passwords. John-MPI passes a password hash to each processor then performed the standard John the Ripper dictionary crack (Pippin et al., 2004).

The BOINC implementation of a distributed password cracker will be structured differently than the two previous implementations. The project will not be using dedicated computers to perform the password checking operation. The project will be using a distributed system for all the modes of John the Ripper (JtR), not just the brute force or dictionary modes. The proper password space break down will be analyzed to effectively take advantage of the structure of John the Ripper and the distributed environment created by BOINC.

E. PURPOSE OF THIS THESIS

This study will help the United States Navy and Naval Postgraduate School by providing a distributed password cracker. It will allow system administrators to better check the passwords of their network computers and will give the Navy the capability to crack passwords of enemy computers. System Administrators can also use it as a training aid to demonstrate the effectiveness of using long, complex passwords and the need for changing passwords at certain intervals. Nation states would have the capability to use a distributed password cracking application against our computers, so using this application against our own passwords will give us the data needed for setting password guidelines in security plans.

THIS PAGE INTENTIONALLY LEFT BLANK

II. BERKELEY OPEN INFRASTRUCTURE FOR NETWORK COMPUTING (BOINC)

A. BACKGROUND

Much of the processing power of the world is now in personal computers. This vast resource of computing power was first tapped into in the mid-1990s by two projects, Distributed.net (2003) and the Great Internet Mersenne Prime Search (GIMPS, 2006). The BOINC projects, that currently take advantage of this public processing power, run on about 1.7 million computers and have the processing capability of about 348 Tflops (Drew, n.d.), calculated from all the countries in the world contributing greater than 500 GFlops[4]. The fastest supercomputer as of late 2007 processes at about 478.2 Tflops (TOP500.org, 2007) and represents good comparision for the distrubited computing power. As of December 2004, more than half of america's homes have high-speed Internet access (Roberts, 2005). The current U.S. population is 304 million (U.S. Census Bureau, n.d.). If one quarter of the U.S. population are home owners, then 76 million homes exist in the U.S. That would mean 38 million computers are available as a resource to any distributed computing project. This does not include the potential from overseas computing power. Clearly one can see that being able to tap into this resource can greatly increase the data processing capability of the scientific community. This resource is now known as Public-Resource computing (Anderson, 2004). Other distributed computing solutions, such as Grid computing, contain supercomputers, clusters, research labs, and other company or government owned equipment with centralized IT professional teams that manage those resources. In these systems personnel must follow rules of conduct. Malicious behavior is treated by removing the person from the company or institution.

Public-resource computing, however, must attract the public to their project to get resources and then keep them interested by offering incentives. The users of those systems are typically not experts and are only interested in advancing scientific research they care about. It is important to note that the public-resource computing area cannot control participants and therefore cannot prevent malicious behavior (Anderson, 2004).

[4] Gflops (gigaflops): A billion floating-point operations per second.

BOINC is a distributed data computation and storage protocol designed to take advantage of public-resource computing. Developed by the Space Sciences Laboratory, University of California at Berkeley, which also created the SETI@home project that performs digital signal processing of radio telescope data from the Arecibo radio observatory (Anderson, 2004). BOINC designers had a few specific goals in mind when they developed BOINC. First, they wanted to "reduce the barriers of entry to public-resource computing." To do this, they made it easy for scientists to create their own projects with as little as a week of initial preparation and only a few hours of maintenance each week after.

Second, they wanted to "share resources among autonomous projects." BOINC-based projects are not registered or centrally controlled by the developers of BOINC. Each project is operated and controlled by that project's designers, which means that they must have their own equipment (Anderson, 2004).

Thirdly, the developers wanted to "support diverse applications" written in several different programming languages that would require little or no modification to work with BOINC. This design criterion is what allows BOINC to be so flexible with the applications it runs.

Lastly, they wanted to "reward participants" who supplied the resources to make these projects work. They did this by creating an "incentive" program which gives participants credits for completing "work units." This generates competition among contributors which would increase the membership of the projects and allow for even more processing (Anderson, 2004).

The use of BOINC as the distribution mechanism for password cracking by John the Ripper did not fit into the mold of the normal BOINC project. John the Ripper is not the classic data processing program that normally utilizes the BOINC structure in that the scientific draw is not present. This project is not curing cancer or searching for the existence of other life. It does verify the strength of the passwords making network security stronger. To this end finding public interest to support the project would be difficult. However, this system works great in a closed environment like a companies' or government's computer system, because system use can be required. Looking back at the

other distributed systems it might not be clear that BOINC is the clear option. The major advantage that BOINC has is that it runs in the background of normal computer operation. So at night when the computers are sitting idle they can be checking the strength of computer system in which is resides.

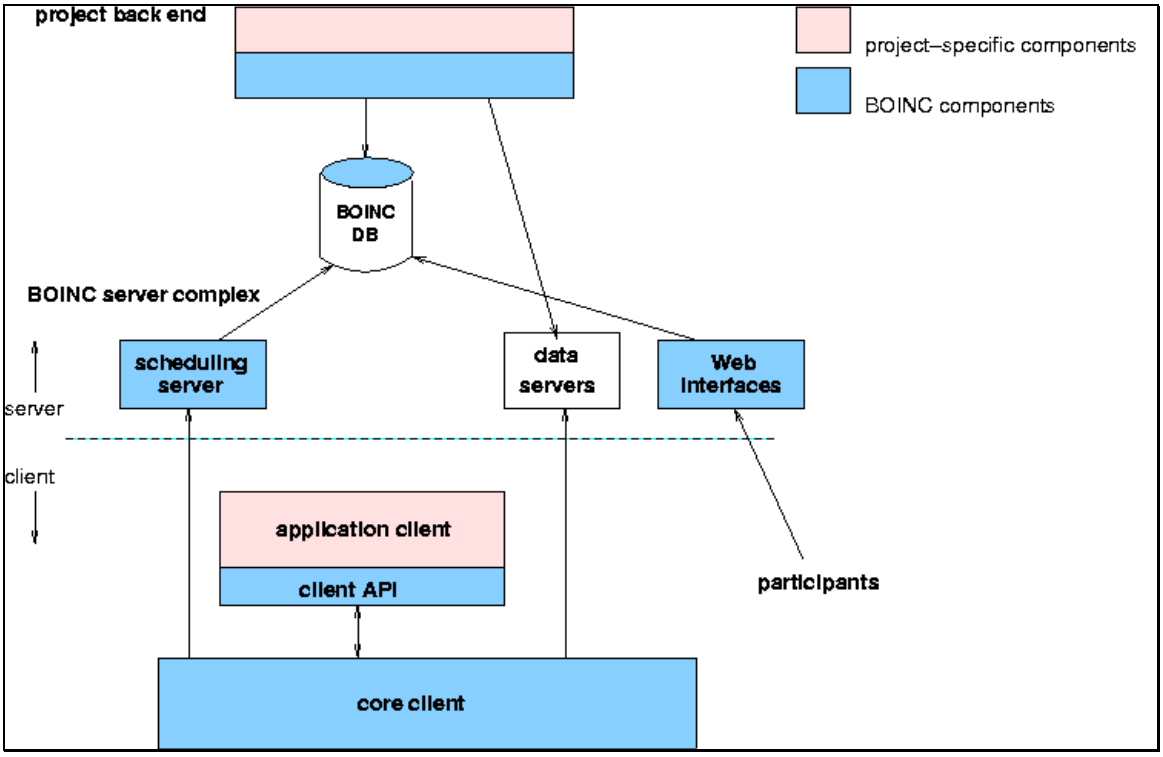

Figure 1. BOINC Project Layout

B. DESIGN

BOINC is a client/server protocol that uses multiple servers to manage application data and transfers. The basic structure of BOINC is a project. This can involve a single application or several applications working together. Scheduling servers control work distribution and report results. Data servers handle file uploads and downloads. It is noted several times that BOINC is designed to be used by scientists not programmers, so tools for controlling the project are simple and well documented. BOINC clients interface with a web server to check on the status of the project and their participation in the project (Anderson, 2004). Below is a detailed breakdown of how each part of a BOINC project works.

1. Server Side

The server side consists of the project back end, BOINC database, and BOINC server complex. Most of these components only need minor or no modification to create a BOINC project. Figure 1 is a graphical representation of a project.

a. Project Backend

The Project Backend consists of the application with BOINC specific APIs, work units, and handles the computational results. The Backend includes the work generator that controls how the work is distributed and is the major application not included with the BOINC basic project backend.

b. BOINC Database

This database maintains all the major data important to the operation of the BOINC project; including platform information, applications running on this project, application versions, user information, host information, work unit information, and results. It is important to note that this is not the science database which would store all data in a form useful to the scientist after the data is processed by the clients. It is important for a project creator to understand that managing the database in conjunction with the work generator is in reality the most difficult part of developing a functional project.

c. Scheduling Server

The scheduling server controls how workloads are distributed to client computers. Clients communicate with the scheduling server when the client needs more work units; the client sustained an error; or the Client returns results. The scheduling server determines how much work to send based on past performance statistics of the client. If a client takes too long, its work units will be classified as lost and sent to another client to be processed.

d. Data Servers

Data servers distribute input and output files through the web server using CGI scripts. The term Data servers is misleading because its actual function is to only

pass data files between the clients and the servers. The data collected in these files is transferred into the scientific database for analysis by other programs.

e. Web Interfaces

The project is managed and accessed through two different Web sites, the project Web site and the administrative Web site.

The project Web site is used by clients to create accounts and join the project. Clients can download the Core Client (see 2.c below), view leader participant's credit boards, or join discussion groups with other members of the project.

The administrative Web site is used for browsing the database, screening user profiles, viewing recent results, browsing stripcharts, browsing log files, and creating user accounts.

2. Client Side

The client side of the project controls all interaction with the client computer. Figure 2 contains a flow chart of the interaction between the Core Client and the server side of the project.

a. Client Application

The client application is designed to be run from the user's computer. This program is written by the project developer to process work units that are passed to it by the scheduling server through the Core Client. It accesses BOINC through client APIs, which communicate with the Core Client.

b. Client API

The Client APIs function as the interface between the client application and the BOINC core client functions. They allow the project developer to access the information passed to the Core Client from the Server Complex.

c. Core Client

The Core Client consists of a collection of finite-state machines (FSM) that handle all the BOINC operations the client needs to perform. FSMs are grouped into containers that manage a particular area of the client program. The core client should never have to be modified. It handles all communication to and from the server. Request for data and other items can be made through the Core Client with the Client API. It is important to understand that the Core Client is downloaded by the users. The users run the Core Client and tell it which project to connect to. That is when the Client Application is downloaded and started by the Core Client.

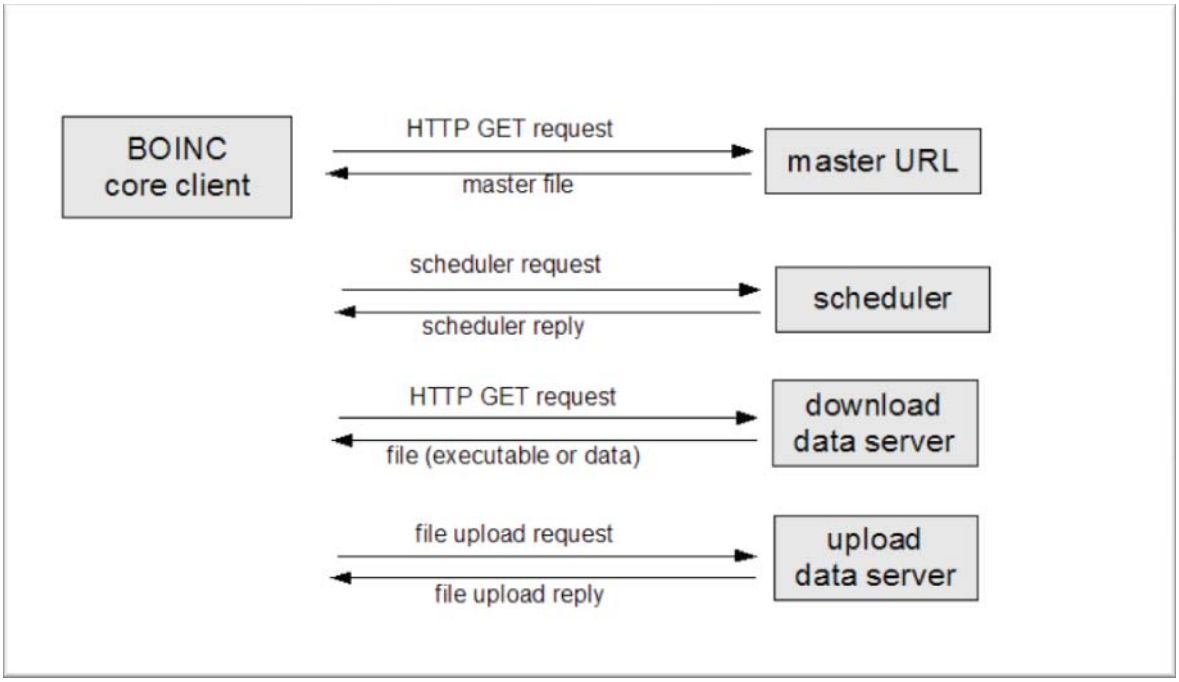

Figure 2. BOINC Network Communication Overview

3. Redundant Computing

Public-resource computing, as discussed before, cannot prevent malicious or erroneous results. Therefore, the developers must have a system for detecting bad results, which in BOINC is called redundant computing. With each work unit that is created for a project, N results are created. Work units are sent to clients to be processed. As the results return they are compared to all the other results associated with the same work unit. Once a threshold of matching results is met for the work unit, the canonical result is

set and all other differing results are discarded. If no canonical result can be found then BOINC server complex creates new results and sends them out until either a maximum result count or a timeout is reached. Only one result can be sent to a participant from a particular work unit and a maximum number of results sent to a participant per day can be set to prevent one person from manipulating the results. This is implemented by using several daemon (server) processes which can be run on different hosts making BOINC very scalable.

It is important to note that when dealing with a password there is only one possible correct answer. This simplifies the verification process and virtually eliminates the need for creating new results. Therefore, detection of false passwords being returned would be very easy if the server checked every password to be correct. This does not cover all cases of misuse; we discuss this further in the Validator section below.

a. Transitioner

The Transitioner implements the redundant computing logic: it generates new results as needed and identifies error conditions.

b. Validator

The Validator examines sets of results and selects canonical results. It includes an application specific result comparison function. For this project, that function does a byte wise comparison of result units. This covers the other case, from above, where a result is returned with no password when a password should be returned. This approach works unless all clients are malicious.

c. Assimilator

The Assimilator handles newly found canonical results. It includes an application specific function which parses the result and inserts it into a science database.

d. File Deleter

The File Deleter erases input and output files from data servers when they are no longer needed. The File Deleter will delete work unit input files when all results are over and delete results output files when the work unit is assimilated, except for canonical result.

4. Failure and Back Off

All client/server connections in the BOINC design use exponential back off in case of failure (Anderson, 2004). Therefore, if the Core Client is not able to contact the server complex or has an error while transferring files it will wait a random amount of time and try again. If that attempt fails, its back off time will be exponentially larger than the first back off time. This helps to prevent crashing a server that just comes back after being shut down for any amount of time.

III. JOHN THE RIPPER

A. BACKGROUND

John the Ripper (JtR) is an open source password cracking utility designed to take a password file; extract the login, hashed password, and associated salt. The program tries several different cracking modes against the hashed password to attempt to break the password. The primary purpose of John the Ripper is to detect weak passwords.

JtR's default configuration auto-detects standard and double-length data encryption standard (DES), Berkeley Software Design, Inc. (BSDI), extended DES-based, MD5-based and Blowfish-based encryption schemes (Openwall Project, n.d.). It also supports Andrew File System (AFS) and WinNT LanMan (LM) hashes for cracking with the addition of a switch at startup (Openwall Project, n.d.). JtR also supports external password cracking by using a limited C based language inside the initialization file. This will cause JtR to use the function(s) defined by a user and compiled by the JtR compiler, for cracking that password. JtR is also optimized for hardware architecture on which it runs including alpha, x86, AIX, HP-UX, and others.

B. DESIGN

1. Basic Structure

John the Ripper (JtR) is divided into two major sections, initialization and cracking. Each section can be broken down into smaller and smaller subsections. This description is a fairly detailed look into how John the Ripper operates, but it is not intended to be a review. The description goes into detail where it will be important for the merging of the distributed protocol and John the Ripper. Several layers of abstraction have been left in place for ease of understanding.

a. Initiation Section

The Initiation section takes in the password file, command line options, configuration file, and loads all encryption algorithms. It then initializes all related

databases used for maintaining and operating JtR. The initialization process can be a complex, but important part of understanding how this program works.

b. Cracking Section

The cracking section determines what cracking mode(s) to use and starts the process for attempting to crack the password. There are five password cracking modes: single, wordlist, incremental, batch, and external.

(1) Single cracking Mode. Single mode takes the login and General Electric Comprehensive Operating System (Wikipedia-GECOS, 2009) field, then processes that information with rules designed to look for simple manipulations to determine the password. The GECOS field usually contains the user's full name and other information. GECOS used to be part of GE's computer division which was bought by Honeywell. The operating system then became known as the General Comprehensive Operating system (GCOS) which was used as print spooling and other services for early UNIX systems at Bell labs. In many UNIX based systems, the field added to 'etc/passwd' file to carry GCOS ID information was called the GECOS field which holds the users full name and other human-ID information. (GCO)

(2) Wordlist cracking Mode. Wordlist mode is similar to single mode, but it reads in a word list instead of the login and GECOS field to crack the password. The list can be compared directly, or as in single mode, a rule can be processed on that word before it is hashed and compared to the password hash. The effectiveness of this attack is dependent on the wordlist that is used.

(3) Incremental crack mode. Incremental mode is a brute force attack on the hashed password. This attack traverses the entire key space using 95 characters and passwords length of 8. The key space is not attacked linearly, but with the use of a character set file which determines how the key space is broken up.

(4) Batch crack mode runs the three previous crack modes one after the other starting with Single and finishing with Incremental. This is the default setting if only the password file is provided. This option was not used in this project.

(5) External crack mode takes code placed in the initiation file, compiles it and then uses that to perform the crack. This option was not used in this project.

2. Data Structures

John the Ripper uses a well organized system of structures to store and transfer data in the program. The type of structure used is a C/C++ object type called a struct. The struct type can contain pointers, variables, methods, or other structs. The implementation of these structures form the internal databases used by functions to process, generate, or store even more data in John the Ripper. Three major databases are used to control and manage the operation of password cracking; the Main Database, the Main Options Database, and the Configuration Database. There are many more structures used in this program. They will all be addressed as they occur. The project utilizes these data structures in the work generator during the initialization phase, but they utilized exclusively in the client application due to remoteness to the scientific data server. The data structures are filled from either a database (in the case of the work generator) or from Core Client application on the user's computer to the client application.

a. Main Database

The Main Database contains data used to track the progression of the cracking process. It is made up of several smaller databases, some of which are created before this database is initialized and then imported into this database. For example, the formats database, which contains all the encryption formats, is brought into the Main Database when a password file is processed.

The struct that forms the Main Database is db_main. It contains several integer variables *loaded, salt_count, password_count,* and *guess_count* to track the state of this database throughout the cracking process.

```
struct db_main {
/* Are hashed passwords loaded into this database? */
      int loaded;
/* Options */
      struct db_options *options;
/* Salt list */
```

```
        struct db_salt *salts;
/* Salt and password hash tables, used while loading */
        struct db_salt **salt_hash;
        struct db_password **password_hash;
/* Cracked passwords */
        struct db_cracked **cracked_hash;
/* Cracked plaintexts list */
        struct list_main *plaintexts;
/* Number of salts, passwords and guesses */
        int salt_count, password_count, guess_count;
/* Ciphertext format */
        struct fmt_main *format;
};
```

(1) Structure db_options. The struct db_options contains three list_main structures, an unsigned integer *flags*, an integer *min_pps*, and *max_pps*. The users, groups, and shells used for a particular password hash in a password file are stored in the three structs *users*, *groups*, and *shells*. Flags for database loading options used in initialization are stored in *flags*. The minimum and maximum passwords per salt are stored in the two integer variables.

```
struct db_options {
/* Contents flags bitmask */
        unsigned int flags;

/* Filters to use while loading */
        struct list_main *users, *groups, *shells;

/* Requested passwords per salt */
        int min_pps, max_pps;
};
```

(2) Structure db_salt. The struct db_salt contains a struct db_salt **next* which points to the next salt in the list. This structure also contains the list of passwords with this salt, and the hash table for this salt or a pointer to the password list. This also contains the integer function call that returns a pointer to the index of the password list to be compared against during the cracking process.

```
struct db_salt {
/* Pointer to next salt in the list */
        struct db_salt *next;
/* Salt in internal representation */
        void *salt;
/* Pointer to a hash function to get the index of password
list to be compared against the crypt_all() method output with given
index. The function always returns zero if there's no hash table for
this salt. */
        int (*index)(int index);
/* List of passwords with this salt */
```

```
            struct db_password *list;
        /* Password hash table for this salt, or a pointer to the
list field */
            struct db_password **hash;
        /* Hash table size code, negative for none */
            int hash_size;
        /* Number of passwords with this salt */
            int count;
        /* Buffered keys, allocated for "single crack" mode only */
            struct db_keys *keys;
        };
```

(3) Structure db_cracked. The struct db_cracked holds the cipher text and plain text of a cracked password and the pointer to the next cipher/plain text pair. It also contains an added variable pwid which is the scientific databases ID for that password on the server side. This functionality was added to simplify the result reporting process from the client application.

```
struct db_cracked {
/* Pointer to next password with the same hash */
      struct db_cracked *next;
/* Data from the pot file */
      char *ciphertext, *plaintext;
      int pwid; //Added for Distributed JtR
};
```

(4) Structure list_main. The list_main structure contains only three entities. An integer called *count* to track the number of items in the singly linked list. Two structures of struct list_entry called *head* and *tail* which are pointers to the beginning and end of the linked list. Each list_entry structure contains a pointer of struct list_entry to the next list_entry object and a character array *data*. This variable is used to store the data in the lists. One example is the list of password files that are passed into John the Ripper at startup.

```
struct list_main {
      struct list_entry *head, *tail;
      int count;
};
```

(5) Structure fmt_main. The struct fmt_main contains four structures. Parameters of the hash function and its cracking algorithm are in fmt_params. Functions to implement a cracking algorithm are stored in the structure fmt_methods. It has a structure for maintaining private variables in the structure fmt_private, which in this case only contains one variable an integer *initialized* to track whether or not this format is

going to be used in password cracking. The last structure in fmt_main is a pointer to the next fmt_main object. The fmt_main structure allows for the creation of a linked list of fmt_main structures which hold all encryption formats possible for decryption. The manner in which JtR handles password cracking formats added a layer of complexity when trying to implement this as a distributed program. It must be assumed that the Work Generator must operate as a server. Therefore it cannot be set for one format once it starts to run. This is discussed further in the implementation sections of the paper.

```
struct fmt_main {
      struct fmt_params params;
      struct fmt_methods methods;
      struct fmt_private private;
      struct fmt_main *next;
};
```

b. Main Options Database

The Main Options Database contains all the information needed to initialize the password cracking process. It is created using the struct options_main which contains the names of files needed for the different cracking modes, flags for operation, a list_main structure (described in section III.B.2.a (4)), and a db_options structure (described in section III.B.2.a.(1)).

The struct list_main *password* stores all the names of the password files loaded from the command line. The struct db_options *loader* contains the flags of all the options used when the main database is loaded as well as other list_main structures. The flags for operation are stored in an unsigned integer *flags*. All the file names are stored in character string variables *format*, *wordlist*, *charset*, and *external*.

```
struct options_main {
/* Option flags */
      opt_flags flags;
/* Password files */
      struct list_main *passwd;
/* Password file loader options */
      struct db_options loader;
/* Session name */
      char *session;
/* Ciphertext format name */
      char *format;
/* Wordlist file name */
      char *wordlist;
/* Charset file name */
      char *charset;
```

```
/* External mode or word filter name */
      char *external;
/* Maximum plaintext length for stdout mode */
      int length;
};
```

There is one more structure that exists underneath the options hierarchy. This structure is used only during the options initialization phase. (See III.C.1.c below) The purpose of this structure is to store all possible command line options and the flags associated with those options. The struct opt_entry contains seven parameters the first is a character string *name* which contains the name of the option. The rest are four flags with type opt_flags (typedef unsigned int), character string *format*, and void pointer *param*. Two of the flags are *flg_set* and *flg_clr*. These flags set the bit that is required for the option and check for duplication. The other two flags *req_set* and *req_clr* used for requirements, meaning that some options require other options. If these flags are set, then the other options are required to be set as well. The requirements for options to be set are based on the operator selected during the command line run. For example if the wordlist cracking mode is selected a dictionary file most also be selected. These requirements are checked at startup, if the options are not set correctly the program will exit and give a message indicating the problem. All the options are stored in an opt_entry array called *opt_list[]*. This array is created in options.c where it statically assigns all values to the array. The void pointer is only used to point to the location of variable stored in another structure. Void is used to allow any datatype to be pointed to.

```
struct opt_entry {
/* Option name, as specified on the command line */
      char *name;
      opt_flags flg_set, flg_clr;
      opt_flags req_set, req_clr;
      char *format;
/* Pointer to buffer where the parameter is to be stored */
      void *param;
};
```

```
static struct opt_entry opt_list[] = {
      {"", FLG_PASSWD, 0, 0, 0, OPT_FMT_ADD_LIST,
&options.passwd},
      {"app",FLG_APP,0,0,0,OPT_FMT_STR_ALLOC, &options.app_name},
      {"single", FLG_SINGLE_SET, FLG_CRACKING_CHK},
      {"wordlist", FLG_WORDLIST_SET, FLG_CRACKING_CHK,
            0, OPT_REQ_PARAM, OPT_FMT_STR_ALLOC,
&options.wordlist},
      {"rules", FLG_RULES, FLG_RULES, FLG_WORDLIST_CHK,
FLG_STDIN_CHK},
      {"incremental", FLG_INC_SET, FLG_CRACKING_CHK,
            0, 0, OPT_FMT_STR_ALLOC, &options.charset},
```

Figure 3. List of Opt Entries

c. Configuration Database

The Configuration Database contains all the data from the John.ini file. This database is created from the struct cfg_section. Each cfg_section structure contains a character string, struct cfg_params **params*, struct cfg_list **list*, and struct cfg_section **next*. The character string *name* contains the name of the section. The variable **name* is a pointer to the next object of cfg_section. All three of these structures are singly linked lists.

```
struct cfg_section {
      struct cfg_section *next;
      char *name;
      struct cfg_param *params;
      struct cfg_list *list;
};
```

(1) Structure cfg_params. This structure contains two character strings and a struct cfg_params. The two character strings hold the name and value of a parameter. The cfg_params structure is a pointer to the next object cfg_params. This linked list holding all the operating parameter passed into it from the configuration file.

```
struct cfg_param {
      struct cfg_param *next;
      char *name, *value;
};
```

(2) Structure cfg_list. This structure contains only two entities both of type struct cfg_line which are pointers to the head and tail of the linked list. The structure cfg_line contains an integer, character string, and struct cfg_line. The integer *number* contains the line number of the string from the file. The character string *data*

contains the text from the line in the file. This is specifically for storing all the rules in the configuration file which tell John the Ripper how to manipulate words from the wordlist or GECOS field information.

```
struct cfg_list {
      struct cfg_line *head, *tail;
};
struct cfg_line {
      struct cfg_line *next;
      char *data;
      int number;
};
```

Falling under the Configuration database hierarchy is the rules preprocessing structure. This structure, defined by struct rpp_context, contains the struct cfg_line **input* and the struct rpp_range *ranges[]*. The structure rpp_context also contains a character string *output* that holds the current rule after preprocessing and an integer *count* which holds the number of character ranges in the rule. The structure rpp_range holds the information about the preprocessed rules.

C. OPERATION

The basic operation of John the Ripper (JtR) is broken up into two major areas initialization and cracking. The program uses command line options to direct the operation of JtR. For JtR to operate correctly, the password file (Figure 6) must be specified. The internals of the program determines the format of file JtR is processing and handles it correctly. Without any other input JtR will automatically run in Batch mode (III.B.1.b(2)). Otherwise, a long list of command line options is available to help the user fine tune JtR's operation.

Just like any other command line program, the operating system passes the full command line string parsed into the program. The main function of this program takes in the first argument from the command line and compares it to four IF statements. If the first argument passed in is "John." Then JtR starts calling the first of three functions *john_init()* followed closely by *john_run()* and finally *john_done()*. The function *john_init()* is passed both the command line argument string and the argument count. This is important because both the work generator and client application are started as

base JtR applications, and then were modified separately. Understanding how this program worked before will make it clear why things were done in the final product described later in this paper.

1. Initialization

Once the initialization function *john_init()* is called, John the Ripper generates all the different structures discussed above. These structures include the different databases and option flag settings. Each structure is either permanent or temporary. The stand alone permanent structures are referenced throughout the entire cracking operation. All the others are created and destroyed as they are needed. In some cases they are created to be moved into a permanent structure. Below will be the full explanation of functions in order of the calls in *john_init()*.

a. Path Initialization - path_init()

The directory from which John was run is the home path of John the Ripper (JtR) for the duration of the cracking process. To be able to find initiation files, word lists, and password files JtR takes the path from the command line arguments and stores it in a character string called *home_path*. Any time a file is opened in this program the *path_expand* function is called. This function takes the file name passed into it and appends it to the end of the *home_path* string. That means all file calls are referenced to that "home" directory.

b. Configuration Initialization – cfg_init()

This section creates the configuration database (III.B.2.c. Configuration Database) from the initialization file john.conf. This file is divided into sections designated by brackets. (See Figure 4.) The sections are options, list single rules, list wordlist rules, different incremental modes, and external cracking modes. The pointer *cfg_database* of type struct cfg_section points to the first section in the initialization file. The database is created one section at a time. Each line of the file is processed by *cfg_process_line()*. This function takes each line of john.conf and determines whether if it is a new section, a parameter, or a string. If the line has an '[' and ']' then *cfg_add_section()* (See (1) below) is called and a new section is created. Else if the

current cfg_section object has its cfg_list structure initialized call *cfg_add_line()*. (See (2) below) Else if line has an "=" in the line call *cfg_add_param()*. (See (3) below) When the end of file is reached the configuration database will contain all the data from the initialization file in a cfg_section linked list pointed to by *cfg_database*.

(1) cfg_add_section(). A new cfg_section object is created. Then the cfg_section object is inserted at the end of the cfg_database by assigning the previous cfg_section object pointer next to point to the current cfg_section object. (Section III.B.2.c) The name of the section is set and the cfg_param pointer is set to NULL. If the name of the section has "list." in it then the cfg_list structure is initialized, otherwise it is set to NULL.

(2) cfg_add_line(). A new cfg_line object is created. Both the line number and data string are stored to the object. Then the cfg_line object is added to the end of the cfg_list for that section.

(3) cfg_add_param(). A new cfg_param object is created. The new object is attached to the end of the cfg_params list. Then the parameter name and value are stored in the object.

```
#
# This file is part of John the Ripper password cracker,
# Copyright (c) 1996-2004 by Solar Designer
#

[Options]
# Wordlist file name, to be used in batch mode
Wordlist = $JOHN/password.lst
# Use idle cycles only
Idle = N

[Boinc]
#Application name
App_name = john_boinc
DB_name = john_db
Host_name =
User_name = john
Pass_word = john
Max_wu = 20
Inc_size = 1000000

Etc…
```

Figure 4. Example John.conf file

c. Options Initialization – opt_init()

This section creates the Main Options database (III.B.2.b. Main Options Database) struct opt_main *options*, which takes in all the command line options. This includes checking for valid and syntactically correct options. When *opt_init()* is called the struct opt_main *options* memory is allocated. The memory for the four list_main structures in *options* is initialized. Once each structure in opt_main is initialized all the command line options are processed one at a time with a call to *opt_process()*. (See (1) below) Then the resulting flags are checked and other flags are set depending on the outcome of the conditional statements. The flags being set are the Main Options database flags that will be used later during the cracking section. At this point the flags that are set are checked again to ensure option dependencies are preserved by calling *opt_check()*. (See (2) below)

(1) *opt_process()*. This function takes the argument list and processes each option in that list with a call to *opt_process_one()*. Then the process one function takes the section of the command line argument (a string) and checks the first character of the string. (See figure 5) If that string begins with a '-', then the option is checked against the full list of possible options. If the option is found, the main options database flags are set to include the flag(s) set by that option as well as any parameters that go with that option. For options that require a file like "wordlist" the storage location is pointed to by the *param* variable of an opt_entry structure. (See III.B.2.b above) If the *param* variable is used it will point to *options*.[variable], which is where all the necessary option file names are stored. If the option does not begin with a '-', then the option is treated as a password file name and stored in *options.password*.

(2) *opt_check()*. This function works like *opt_process()*, but it checks the required flags instead of the operating flags to ensure option dependencies are correct

```
[]# ./john --single --wordlist=Wordlist.txt –rules  passwordfile.txt
```

Figure 5. Example Argument List

d. Format Initialization – john_register_all()

This section forms the structure fmt_main stored in the struct fmt_main *fmt_list*. (See III.B.2.a(5) above) Each encryption format is loaded one at a time creating a new fmt_main for each format, then that fmt_main object is added to the end of the *fmt_list* chain of fmt_main objects.

e. Main Database Load – john_load()

The section loads the Main Database based on the flags set in the Main Options Database. Some of the options that can be set change the way this database is loaded and change operation of the John the Ripper. This section is divided up into different if-then blocks of code. The order of the block is significant. I am only going to discuss the if-then block that pertains to password cracking. That block is characterized by options that require a password file. Once inside that block of code the first option that is checked for is the "show" option. If this flag is set then John the Ripper reports what it has cracked to the screen and exits the program. The next option that is checked is "single" or "batch." If this flag is set then *options.loader.flags* is set to tell the database loader that the login and GECOS information needs to be loaded. After this the Main Database, struct db_main *database*, is initialized by calling *ldr_init_database()*. (See (1) below) Each password file name is then passed into *ldr_load_pw_file()*. (See (2) below) Following this the file containing previously cracked passwords is loaded into *database* by calling *ldr_load_pot_file()*. The *loaded* variable in the database is now set to one. Finally, information about how many passwords are loaded, how many different salts, and what format is going to be used to crack the password is sent to the console.

(1) *ldr_init_database()*. This function takes in *options.loader*, a db_options object, and stores that data into *database.options*. (See III.B.2.a above) The *database* object then has all other variablesinitialized.

(2) *ldr_load_pw_file().* Once this function is called, it immediately calls a function *read_file()* which takes a function *ldr_load_pw_line()* as one of its arguments. The *read_file()* function takes the filename passed to it, expands the path (See III.C.1.a above), and opens the file. Once the file is open it loads one line at a time into the program. Each line is passed to the function as an argument. In this case it is the

ldr_load_pw_line() function. This function takes each line of the file and parses it according to the structure of a password file, which is determined by the encryption scheme used by the computer from where the password file was retrieved. Each line of a password file has the structure: login, hashed password, UID, GID, [Full name], home directory, and shell as shown in Figure 6. Every line of data is stored as an All this data is stored in its proper location.

```
root:$1$DkqWcGo0$.6Z.dgYnfusOljL.c0Kz41:0:0:root:/root:/bin/bash
bin:*:1:1:bin:/bin:/sbin/nologin
test1:$1$9HWiP8Lm$extV0fqcQBJhSAKMssbPt.:501:501:maderight:/home/test1
:/bin/bash
test2:$1$hNv/cSMI$HTHfQUfMgZ3/lKUUNC5il1:502:501:test2:/home/test2:/bi
n/bash
test3:$1$xLE3ztN3$KonehXd4nGnFLnJhiWCxG1:503:100:test3:/home/test3:/bi
n/bash
```

Figure 6. Example Password File

2. Cracking

The cracking process is started when the function *john_run()* is called. This function does nothing more than run through a sequence of if-then-else statements to determine proper operation of this iteration of John the Ripper. Again *options.flags* is used to determine what options were set at the command line.

Password cracking is the only option to be discussed in this study. These consist of all five cracking modes: single, wordlist, incremental, external, and batch. All the cracking modes are independent and self-sufficient except for the batch cracking mode. (See III.B.1.b(4) above) There are some similarities between the cracking modes, but all are different as previously discussed.

a. Single Cracking Mode

The function *do_single_crack()* is called from either *john_run()* or from the batch cracking mode. This function starts by defining a rules preprocessing struct rpp_context **ctx*. Once that is completed the single crack mode goes into the function

single_init(). This function gets all the variables and structures setup to run a single crack mode run. A function called *single_count_rules()* is run inside *single_init()*. The function *single_count_rules()* sets the pointer *ctx* to the head of the rules list that resides in the "single" section of the *cfg_database* and it initializes in memory how the rules are processed. Then it returns a count of how many rules exist for the single cracking mode section. Once the initialization of the single cracking mode is complete, function *single_run()* is called. The function takes each rule for single crack mode and applies it to each word of the login and GECOS data in turn. Each time a rule changes a word, that processed word is run through a hash function and compared to each of the hashed passwords stored in the hashed passwords data structure. If there is a match then that password and hash pair are stored in the Main Database under *cracked_hash*. If no match is found, the memory for the single crack pass is de-allocated. John the Ripper exits telling the user how much time was spent trying to crack the passwords. All found passwords would be reported in addition to the total not found.

b. Word List Cracking Mode

The function *do_wordlist_crack()* is called from either *john_run()* or from the batch cracking mode. It is similar to single cracking mode in that it can processes words with rules, but the structure of the function is different. The wordlist file is opened, then the function *wordlist_count_rules()* is called. This function operates the same as *single_count_rules()*. Except this time it retrieves the section called "wordlist." The number of rules and the pointer to the rules list are stored as an object. Now the first rule is selected and every word from the wordlist file will have that rule applied to it. An example of rule would be, replace all a's with the @ symbol in every word for which this rule is applied. After the rule is applied to a word, it is passed into a hash function and compared to the hashed passwords that are being cracked. If there is a match, that password and hash are stored in the Main Database. If not, the next word will be checked until there are no more rules or no more hashes to crack. John the Ripper will exit as described above.

c. Incremental Cracking Mode

The function *do_incremental_crack()* is called from either *john_run()* or from the batch cracking mode. The incremental mode starts by accessing the *cfg_database*. It finds the "incremental" section with the subsection name "All" from that database and pulls out the "File" parameter. That parameter is the name of the file that holds the character set for this run. It then puts in the rest of the parameters for that section. If hashed passwords were LanMan hashs then the subsection name would be "LanMan." The character set file is now opened, the header is pulled in, and file is checked for format errors. All memory allocation is now complete. At this point John the Ripper starts going through the entire key space checking every possible combination of words from length one to length eight. Just like the other cracking modes above JtR will inform the user when it is done and if it has cracked any of the hashes. JtR traverses the key space in a non-linear fashion. The order of the traversal is determined by the character set file. The character set file is a binary file broken up into smaller segments. Each segment has three variables *length*, *fixed*, and *count* which determine how many different guesses are in that section.

The variable *length* signifies the actual length of the password guess being checked plus one. So if *length* equals *1* the length checked would be two and only letter combinations of length two will be checked in this section. The length is used to determine a specific location in the binary file to start loading data structures with strings of characters in varying sizes. For illustration the following would be example strings: *UIOisru, &*(aliurnw, abcdefABCDEF, or 1. The only rule is that there are no repeats in any string. The designers of JtR had a process for deciding how to store the strings in this manner, but exact process was not part of this research. It is important to understand that this data structure changes every time the *length* variable is different than the previously worked section.

The variable *fixed* signifies the number of digits of the guess that will remain fixed for this section plus 1. Therefore, if the length was five and fixed was two,

the first three letters of this section's run will stay the same. Examples would be: strean, stral2, and strae4. All guesses are six characters long (*length* + 1) and the first three characters remain the same.

The variable *count* signifies the number of different fixed letters combinations a section's run will have plus one. Therefore, if the length equals five, fixed equals two, and count equals one; then this section will have two different fixed letter combinations, all guesses being six characters long. The result would be two groups of three fixed letter passwords all six characters long. The following would be good examples: strean, stral2, and strae4; and grilet, gri234, and grills.

The example sets are small for a reason; the actual size of each section can range from one guess to 2E15 guesses. This simple fact has a major effect on distribution of work in this distributed application as discussed later in the paper.

d. Batch Cracking Mode

The function *do_batch_crack()* is only called from *john_run()*. Batch cracking mode runs single cracking mode, word list cracking mode and incremental cracking mode in their default settings to crack a password file. It does this by calling first *do_single_crack()*, then *do_wordlist_crack()*, and finally *do_incremental_crack()*, allowing each crack type to run its course on a password file. This mode was not used in the BOINC adaptation of this program for reasons discussed below.

THIS PAGE INTENTIONALLY LEFT BLANK

IV. MERGING BOINC AND JOHN THE RIPPER

A. ISSUES MERGING BOINC AND JOHN THE RIPPER

To understand how to merge BOINC and John the Ripper together there must be an understanding of what BOINC needs to properly distribute the task that JtR performs. BOINC calls for the creation of two programs (Work Generator and Client Application; both discussed above) and several project specific functions to be called by the Assimilator and Validator. The Work Generator, Assimilator, and Validator all operate as daemons that communicate through the BOINC database. They all perform operations on the data in the database using queries to separate the data being retrieved.

Beyond the need to understand what BOINC needs to operate there is a need to ensure JtR does not lose its efficiency and that proper password tracking occurs. JtR was designed as a linear program; it is started with a password file or group of password files, and then processes all the passwords until finished. To operate as a work generator, JtR will need to operate as server; generating work units when they are needed and having the ability to add passwords to the cracking process as they are added to the system. What will a work unit consist of? Will it be more efficient for one password per work unit or multiple passwords? How are groups of passwords tracked in the cracking process? How is a newly added password separated from a group that is further along in the cracking process? These questions are answered in the development of the different programs. Each program and function will be individually discussed in detail below.

B. WORK GENERATOR

The Work Generator is the back-bone of the entire project. The work units that generated by the Work Generator are the means for the distribution of the password key space. It is designed to be a separate daemon that can run on a machine physically separate from the Validator or databases if the load of the project is too great.

1. Planning

The plan was to take the normal development of JtR and convert into a Work Generator. As previously discussed, JtR has two distinct sections: initialization and

cracking. The Work Generator mainly needs the initialization section, but does use the first part of the cracking section. As discussed above JtR has three main cracking modes. Each cracking mode is considerably different requiring a different method to distribute the work of each mode.

The original plan for the Work Generator was to, at start up, feed a password file into the program from the command line which would load the passwords to be cracked into the JtR Database. Once the passwords were in the JtR database, they would be tracked all the way through the cracking process. To add new password files the Work Generator would have to be stopped and restarted with a new file noted on the command line. The JtR database would maintain the separation required to track the password crack history. When the passwords from the password files are loaded into the database, the work generator would begin creating work units.

Instead, the job of loading the passwords into the JtR database was assigned to a new program specifically for loading passwords, this allows the Work Generator to continue work generation without stopping.

2. Implementation

a. Single Crack Mode

Single crack mode can be completed by a single computer in a matter of seconds. Therefore, Single crack mode will be a single work unit. A client computer will take the GECOS information passed into it and process that information with all of the rules.

b. Wordlist Crack Mode

Wordlist mode which works similarly to Single mode cannot be easily done by one computer if the dictionary used is large. Since the Wordlist mode uses one rule and applies it to every word in the dictionary the simplest breakdown of work units is by rule number. As long as all computers work with the same rule list, a rule number can be passed to a client computer which can find the rule required. Once the correct rule is found, it can process the dictionary with the rule and check the password hashes passed to the client.

c. Incremental Crack Mode

The Incremental mode is the most difficult to distribute. As discussed above, JtR traverses the password key space by segments. The original design of this program used this fact alone to distribute work. The segments and order of those segments where determined by the character set file. Again, since the same character set file is passed to all clients, each segment can be given a number and that number can be passed to the client. Once the client has the number it can find the place in the character set file and start cracking that segment of the key space. The idea is very simple and works very well with the exception of the segment which processes 2E15 guesses. The problem with this method of work distribution will be discussed in the troubles section below along with the restructuring to the entire Work Generator and required changes to all other programs because of this restructuring.

d. Control and General Structure

No matter what cracking mode is to be used, all three cracking modes abstractly work the same. The commonality between the modes is a reference number. For Wordlist mode this means a rule number, but for Incremental mode this means a location in the key space. Therefore, all crack modes can call a single function that will take all the information from the JtR database and generate work units. The idea of a reference number is still used for Single and Wordlist Crack modes, but new structure needed to be created for the Incremental Crack mode.

e. Basic Scheduling

This password cracker is designed to take multiple password files and store them in the JtR database. Then sort through the passwords grouping them by priority, format, and location in the cracking process. It is clear that a password file that started cracking a month ago will be at a different point in the cracking process than a password file added today. However, users of the system do not want to have to wait on the Incremental cracking mode to be completed for one batch of passwords before the single or wordlist cracking modes can be run on the newest passwords. Therefore, a hierarchical system for work generation was created so that all passwords in the system to

be cracked will be checked in Single crack mode before they are checked in Wordlist. It follows that all passwords will be checked in Wordlist mode, before starting Incremental mode. This is a very simple scheduling problem that the shortest jobs should always be run before longer jobs to increase throughput. Starvation of the longer jobs is a concern when attempting to crack a strong password, but the main purpose of this cracker is to catch weak passwords. Therefore, any passwords that have not been cracked by the time Incremental mode is started can be considered relatively strong passwords.

f. Priority

Priority in this program is the number of work units generated at one time. Since work units can be generated very quickly it is important that all passwords, or password groups, get equal treatment. Therefore, all password groups get a set number of work units generated per cycle so each group progresses at about the same rate. If there is a desire to speed up the process on one or more passwords, then their priority can be raised. The selected passwords will generate more work units per cycle, therefore causing them to be processed faster.

g. Format and the Batch Concept

Format is a very important grouping. Different operating systems store passwords in different ways. For instance, the Linux operating system uses FreeBSD MD5 while older windows operating use Landmann hashes. JtR is only designed to crack one format at time. This makes sense because operating systems just use one type of password storage system. Therefore, each password file could have a different format, but each password in the file will have the same format. This means that differently formatted passwords will have to be grouped separately and cracked separately. Therefore, the idea of a batch was created to separate passwords into different groups for work distribution. The original implementation of the batch concept was to generate a batch “on the fly” by loading all the passwords that were not cracked into a list. Then sort the list and subdivide until all the passwords where grouped by format, priority, and location in the cracking process (ruleid). The database would then check for a batch that matched those criteria and recover the information. The batch was stored in the database

as a batch id, priority, format, and ruleid. If no batch existed, one was created. The ultimate purpose of the batch was to ensure the work unit name and files were unique. The batch id was added to the file names and work unit name to ensure that uniqueness was maintained at all times. The change in the Incremental Mode key space division required a major change to how batches operate, which resulted in a simplified process for grouping passwords in batches. This change is discussed in the troubles section.

h. General Work Generation Sequence

The original operation of the work generation step was to check the BOINC database to see if a set number of results had been created and sent to a client to be processed. Once this criteria had been met it would hop to a set of "if then else" statements. Each IF statement checked the JtR Database password status *mode_flag* variable to check for passwords being processed in Single, Wordlist, or Incremental Crack modes. The following IF statements checked first for Single, second for Wordlist, and thirdly for Incremental Crack mode. If passwords in the JtR Database were required to be cracked in the Single Crack mode then the function do_single_crack() would be run. If passwords were required to be cracked in the Wordlist Crack mode then the function do_wordlist_crack() would be run. Once all the passwords in the JtR Database were verified not to be able to be crack by the first two methods, only then would the Incremental Crack mode function do_incremental_crack() be run. When no passwords remained to be cracked the program would exit.

Each of the do_X_crack() function calls work slightly different, but essentially the same. Once run the function determines the max number of work units that mode could produce. It would then call gen_work() with arguments the mode of cracking and the max number of work units possible.

Once inside gen_work() all the passwords to be cracked in that cracking mode would be loaded into a vector. This vector would be sorted first by format, storing the different format types in a vector. A loop would then be run, stopping when vector containing the different formats was empty. One of the formats would be removed from the vector of formats. The passwords that matched that format would be moved into a different vector storing passwords.

This process was repeated for priority and ruleid each having their own loop. Once inside the ruleid loop a check for batch would be called. When the batch information is returned the final loop is started. This loop will terminate when the number of work units created is equal to the priority of the batch. Inside this loop a function called gen_files() is called passing in the batch id and the vector of passwords.

Gen_files() creates three things: a password file, a crack file, and a work unit. The password file is created and stored in a location organized by the hash of the file name. This is part of the BOINC functionality which improves efficiency by reducing the search time for a file. The function called create_pw_file() takes in the vector of passwords and a file pointer to write all the passwords in a file that looks identical to a normal password file. The name of the file is "pw_batch id_rule id.pw."

The crack file is created and stored in a location BOINC recognizes. The function create_cr_file() takes in the cracking mode, the vector of passwords, and a file pointer. It writes to a file all the information that the client application will need to run. This will be described in more detail in the Client Application section.

Finally, the function create_work() is called. This function takes in all the files required for the client application to work and a unique name then generates a work unit. This function is a BOINC API function which interfaces with the BOINC database and creates the work unit and all the required results to be processed by clients.

For completeness, what was just discussed was the original structure of the Work Generator. Below will describe the changes that occurred and why those changes were required.

3. Troubles and Changes to Initial Plan

a. Incremental Crack Mode Problems

The reason for all the changes in the JtR program comes from understanding the problem with the method of distributing the password key space in Incremental Crack mode. At the start of this research it was not clear how the brute force cracking process worked. Precisely how the key space was traversed was not completely understood. It became clear that not all segments in the Incremental Crack mode look at

the same amount of key space. The segment size of length 3, fixed 0, count 0, is only one word. However, the size of the segment that has length 7, fixed 7 and count 27 is 1.8 billion words. If a computer can check 1000 passwords per second it would take 20.8 days to process that one work unit which is not even the largest work unit. Clearly this is too much time for one computer to be processing one work unit. Because of this elongated time table, it was determined that this approach was not feasible.

The solution turned out to be simpler than expected, but required greater knowledge of how the key space is traversed. The solution was to standardize the size of the work unit by counting out the number of passwords that one computer could check in a reasonable amount of time. The JtR database had to be modified to track the starting and ending position of that count and translate this information to work generator functions. The inputs to the gen_work() function call for the Incremental Crack mode became significantly different from those of the Single and Wordlist Crack modes. It became clear that a batch driven work generation process would be much more efficient and a lot less cumbersome than the previous implementation. A batch object is now the base object that is passed between functions that are part of the work generation process. Passwords are no longer all pulled out of the John Database and sorted, they are now pulled out based on their batch id and status.

b. Batch Concept

The Batch concept in the corrected version of the Work Generator is to organize the passwords into groups and track them for the entire cracking process. Batch numbers are assigned to passwords immediately after the passwords are loaded into the JtR database. The batches in the database are searched to find a match for the passwords that were just stored. The search is based on the format of the hashes and if passwords are WORKING[5] or not. A password is WORKING if work units have been created for that password. If a match is found and that batch is not WORKING; the new passwords are given that batch id. If a match is not found or that batch is working, a new batch is

[5] WORKING is a #define integer that signifies that batch is in the password cracking process.

created and the passwords are given the batch id of the new batch. In addition if a password hash from the password file already exists in the JtR database then it is not added to the database.

With the change of the batch concept, the information it stores also changed. The batch no longer contains a ruleid or format. The format is checked when the passwords are loaded and at no other time because the password's format does not change. Ruleid is replaced with start location and other variables added specifically for to control how the Incremental Crack mode is implemented. Single and Wordlist Crack modes have not changed, but the start location is used in place of ruleid. The batch object now contains the current crack mode of that batch.

There is no program functionality that supports the changing of password priority. Priority of batches can be changed, but only manually in the JtR Database. Individual password priority would have to be changed manually and a new batch created to hold it, if this is desired. Future changes to this implantation could incorporate this change.

During the work generation loop all the batches are loaded into a list. If there are any batches that are in Single or Wordlist Crack modes, the batches that are in Incremental Crack mode will be removed from the list. This ensures all batches not at the Incremental Crack mode will get processed prior to those in the Incremental Crack mode. Every batch object will have work units produced equal to the priority of that batch. Once all the batches are processed then the loop starts all over again.

For example, if JtR Database has three batches, the first batch is a LanMan format in the Incremental Crack mode, the second is a MD5 format in the Incremental Crack mode, and the third is a MD5 format in the Wordlist Crack mode. The first and second batches would not get processed for work units until the thirds batch has reached the Incremental Crack mode. Therefore, once the third batch reaches the Incremental Crack mode each batch will get work units equal to their priority per cycle of the work generation loop.

Since the focus of this project is to locate weak passwords the starvation of batches in the Incremental mode is allowed so that new batches can be checked for weak passwords via Single and Wordlist Cracking modes.

c. The Password Loader

The password loader has two purposes: first to load password files into the JtR database and secondly, to organize the passwords in the database into batches. JtR is designed to check each guess against every password hash passed into the program. Therefore, leaving the password file in a whole unit is the best option for maintaining the efficiency of the crack. The password loader takes this one step further. After password files are stored in the database the other passwords in the database are checked for matching hash format and whether or not those passwords have been worked. The meaning of "worked" used here is to reference if the password has had work units created. Both criteria are required because JtR only works on one hash format at a time and new passwords cannot go into older batches without skipping a large section of their key space. If the formats match and passwords in the database have not been worked, then the new passwords are added to a batch that is already present.

The Work Generator can be running during this process or not. If it is currently working on a batch of passwords, it will generate work for the new batch on the next cycle. If the Work Generator is not running, then the passwords will be ready once it is started. The Password loader will create connections to both the JtR database and the BOINC database. With those connections it can retrieve and updated password information and retrieve the status of current work units.

The Password loader is currently designed to be run each time a new password file needs to be loaded. In the future it could be used as a daemon for password files to be loaded at anytime. That change is beyond the focus of this project.

d. New General Work Generation Sequence

The new work generation sequence starts very similarly to the previous sequence. A while loop checks the John database to make sure passwords exist that need to have work generated. Then an IF statement containing the function check_ready_wu()

is called. This function queries the BOINC database twice. The first is to count the total number of work units stored in total_wu_count. The second is to count the number of results that have produced a canonical result stored in total_wu_processed. If total_wu_count <= total_wu_processed then a true value is returned to the parent function. That will enable process_wu_by_batch() to be called. If check_ready_wu becomes false, the Work Generator will enter a sleep mode.

The function process_wu_by_batch() will cycle through all the batches generating work for all the batches based on the priority they have set, default is 20. The function starts out by loading a vector with all of the batches in the JtR database that are not complete. Complete in this context indicates that all passwords in this batch have been cracked. Once all the batches have been loaded the vector of batches is cleaned. This is the process of removing batches that are in the Incremental Crack mode if batches exist that are in the Single or Wordlist Crack modes. After the vector has been cleaned a batch is removed and stored in a new batch object. This batch object is used to control the entire work generation process for this batch. A while loop is started that tracks the number of work units created and exits when that number reaches the batch priority.

Inside that while loop is the load_jtr_db_formats() function which takes in a struct db_main *database(see above) and the batch being processed. The function queries the JtR Database for the format of the passwords in this batch. That format is matched with a format loaded into the struct fmt_main *fmt_list which is then assigned to the database->format. This information is needed in two of the cracking modes to verify correct processing for that work unit.

The decision for which crack mode to perform is implemented with an IF THEN ELSE chain. Each IF statement checks the batch mode_flag then performs the correct work generation crack. Only one work unit is generated on each pass. After each run, the current batch information is updated in the JtR database. Each function call do_*mode*_crack() below takes a db_main object and a batch object.

If the batch mode_flag is SINGLE then do_single_crack() is called. This function calls gen_work() and checks for proper work generation. It then changes the mode_flag to WORDLIST and exits.

If the batch mode_flag is WORDLIST then do_wordlist_crack() is called. This function loads all the rules from the wordlist subsection of the configure file, stored in the cfg_database at program startup. Once this is complete the start location (s_loc) of the batch is stored in recovery rule number (rec_rule). The function restore_rule_number() is called. This function searches for the rule that corresponds to the rule number stored in s_loc. The start location holds the value of the next rule to be processed. Once the rule has been found the rule is verified to be good. If it is not good the rule number increments, the start location is incremented and the next rule is checked. If it is good then gen_work() is called. When a work unit has been created correctly generated_work gets set to one. The next thing to be checked is if the next rule is the end of the rule list. If it is then mode_flag gets set to INCREMENTAL, s_loc is set to zero, and the loop is stopped. If the next rule is not the end of the rule list then rule number is incremented and stored in the start location. Since the work unit was generated do_wordlist_crack() is exited.

If the batch mode_flag is INCREMENTAL the do_incremental_crack() is called. This function pulls parameters from the cfg_database like character set file name, minimum password length, maximum password length, maximum character count, and incremental work unit size. Memory is then allocated for all the arrays used to control word generation. The batch object passed as an argument for the do_incremental_crack() function contains the final location, length, fixed, count, and all final numbers from the previous batch run. Those items are stored in the start location, length, fixed, count, and start numbers of the current batch object. Now the start location is stored in recovery entry and all the start numbers are stored in the recovery numbers array locations. At this point, the order pointer is set to the start location and a loop starts that will not stop until the last location in the order is reached. Every pass of the loop works through one segment of the key space. All the arrays that control word generation now are filled from the character set file. Once the memory has been loaded inc_key_loop() is called. Each word that is generated increments the word counter. The word counter variable is a global variable so several segments can be processed in one run of do_incremental_crack(). When the word counter becomes equal to the incremental work unit size, inc_key_loop() returns a one.

The default value for incremental work unit size is one million. This figure is based on a computer being able to process one thousand passwords in a second. With this assumption it would take sixteen minutes for that work unit to be processed.

With a one being returned from inc_key_loop() all batch final information is updated. The current entry is stored in final location, length, fixed, count, and the numbers array section of the batch object. At this point gen_work() is called, if a work unit is created correctly the loops is exited and do_incremental_crack() is exited.

The function gen_work() was simplified significantly with the changes in the structure. This function takes a batch and loads a vector of passwords that have the same batch id as the current batch. Once the vector is loaded, it and the batch object are passed to gen_files(). The function gen_files() did not change with the exception of the naming convention for the password files, crack files, and work unit name. Password files now have the convention "pw_ batch id_ mode flag_ start location_ concatenated start numbers.pw." Crack files are similar except they start and end with "cr" instead of "pw" and the work unit name is the same as the password file name except no file extension, ."pw" at the end.

e. Server Logic and BOINC Interfacing

When the BOINC server complex is started up it individually starts each daemon (server) and controls access and scheduling of the daemons. Function calls are required to store created files in a location the BOINC data server can find. During program cycle a function call is required to determine if a stop command had been given, which requires prompt exiting of each daemon. The Work Generator is the daemon that was required to be generated completely without any template or guide from BOINC. However, the Work Generator is a subordinate daemon which must work with the controlling server complex. BOINC made integrating the Work Generator daemon into the server complex simple. Function calls are included with BOINC for storing output files in the correct location, connecting to the BOINC database, and a function call to determine if the Work Generator needed to exit. The addition of these BOINC function calls aided in the shifting of JtR to a server application.

The Work Generator required its design and implementation to be able to deal with new passwords being added while running. Sort out the order of the new batches and even stop work on older batches if they are in the Incremental Cracking mode when a new batch is added. All these design requirements have been included in the Work Generator. Figure 7 shows the flow diagram for the Project Backend and how the Work Generator was integrated. It also shows the relationship of the John the Ripper DB (discussed in the next section) and Password Loader to the Project Backend.

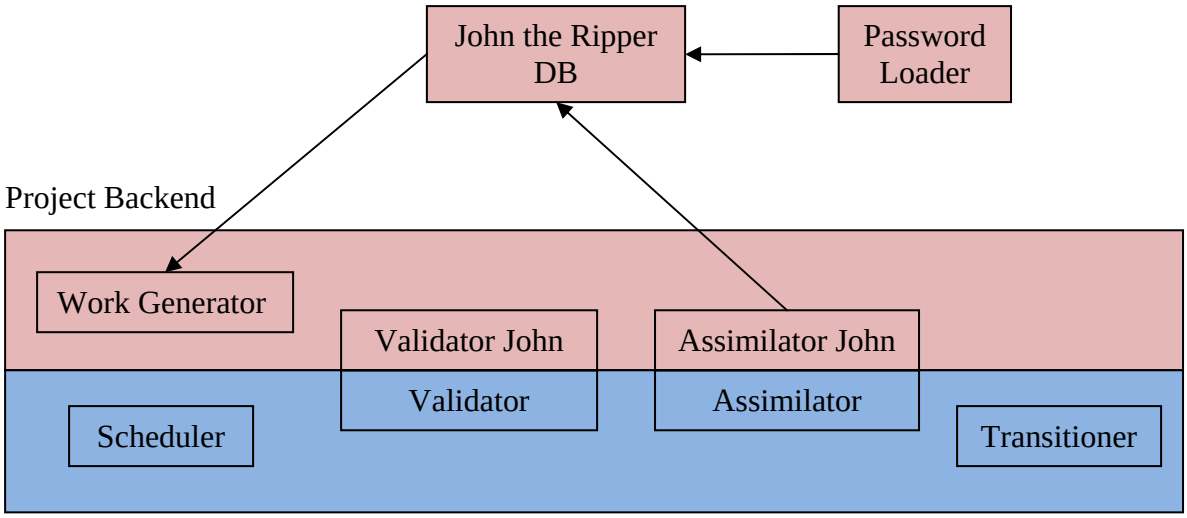

Figure 7. John merged with BOINC – Server side

C. JOHN THE RIPPER DATABASE

1. Construction

The general design of the John the Ripper (JtR) Database is very simple. It consists of three tables: passwords, status, and batch in a MySQL database. The passwords table contains all the fields of a password stored in a password file plus batch id. Every password has one batch id, but many passwords have the same batch id. The batch id, as discussed earlier, is used to group and track passwords throughout the cracking process.

The status table contains the password id, status flag, and priority of the password the status object is assigned too. Each status has one and only one password. The status tracks the general progress of its password through the cracking process, as well as the number of work units that have been created and completed for this password.

The batch table contains all the information required to track a batch's progress through the work generation and cracking process. Including the current batch cracking mode, the priority of the batch, and the information required to create the next work unit.

a. Schema

```
create table  passwords(
    id          integer     not null auto_increment,
    login         varchar(254) not null,
    passwd_hashvarchar(254) not null,
    userid        integer default 0,
    groupid       integer default 0,
    GECOS         varchar(254),
    home_dir            varchar(254),
    shell         varchar(254),
    password            varchar(254),
    format        varchar(254),
    batch_id    integer default -1,
    primary key(id)
) type=MyISAM;


create table status(
    id            integer not null auto_increment,
    pwid    integer not null,
    status_flag   integer default 0,
    priority      integer default 0,
    workunits_gen bigint default 0,
    workunits_done  bigint default 0,
    primary key(id)
) type=MyISAM;


create table batch(
```

```
    id            integer not null auto_increment,
    mode_flag     integer default 0,
    batch_priority    integer default 0,
    s_loc integer default 0,
    s_length integer default 0,
    s_fixed integer default 0,
    s_count integer default 0,
    s_num0 integer default 0,
    s_num1 integer default 0,
    s_num2 integer default 0,
    s_num3 integer default 0,
    s_num4 integer default 0,
    s_num5 integer default 0,
    s_num6 integer default 0,
    s_num7 integer default 0,
    f_loc integer default 0,
    f_length integer default 0,
    f_fixed integer default 0,
    f_count integer default 0,
    f_num0 integer default 0,
    f_num1 integer default 0,
    f_num2 integer default 0,
    f_num3 integer default 0,
    f_num4 integer default 0,
    f_num5 integer default 0,
    f_num6 integer default 0,
    f_num7 integer default 0,
    primary key(id)
) type=MyISAM;
```

b. Software Access

Software access is controlled and structured through structs and classes coded in C++. The file john_db.h/C contains all the structs and classes, except for the struct BATCH. Since C can be compiled in C or C++, but C++ cannot be compiled in C that struct was placed in boinc.h because BATCH was used in C parts of the code as well as C++ parts of the code.

Any function that was required for C parts of the program are written in boinc.h/C. The file boinc.h is complied as a C header file. All interfaces are written for C code, but the internals of all the functions in boinc.C are written in C++. John the Ripper is written in C while BOINC is written in C++. Therefore, BOINC required calls had to be complied in C++. Because of this unique structure of the program all parts of the JtR program that required access to the John database had to call interface functions through boinc.h. The only file in the Work Generator that calls john_db.h functions is boinc.C. The Assimilator is the only daemon outside of the Work Generator that calls john_db functions. Therefore, john_db.h/C are written in C++.

The base internal structure matches the schema variable for variable using structs defined in john_db.h. Below is the implementation of the database interface.

(1) Password structure. This data structure is designed to hold a password as an object PASSWD. It is part of several other data structures.

```
struct PASSWD {
    int id;
    char *login;
    char *passwd_hash;
    int UID;
    int GID;
    char *GECOS;
    char *home_dir;
    char *shell;
    char *password;
    char *format;
    int batch_id;

    void clear();
};
```

(2) Status structure. This data structure is designed to hold the status of a password as an object STATUS.

```
struct STATUS{
    int id;
    int pwid;
    int status_flag;
    int priority;
    unsigned long workunits_gen;
    unsigned long workunits_done;

    void clear();
};
```

(3) Password Item structure. This data structure is a combination of both the password structure and status structure as one unit for ease of manipulation and referencing.

```
struct PASSWD_ITEM {
      PASSWD pw;
      STATUS pw_stat;

      void clear();
      void parse(MYSQL_ROW&);
    PASSWD_ITEM& operator=(const PASSWD_ITEM &b);
};
```

(4) Class DB_PASSWD. This class is designed to be the direct interface between the PASSWD struct and the passwords table. It is a sub-class of DB_BASE which is a defined class of the BOINC database. That gives this class access to a DB_CONN variable, which is the established connection to the john_db database on the mysql server. This connection is setup during the initialization of the Work Generator. Class DB_BASE has three virtual functions: get_id() returns the id of the object, db_print() prints the contents of the PASSWD object to a buffer, and db_parse() which loads the PASSWD object from a query on john_db. The check_dup() function queries the password table for passwords with the same login, password hash, and format. The load_pw() function is called from the password loader when a password pulled in from a password file is ready to be stored in john_db. This function stores all the information about the password in the PASSWD object's fields then calls insert() which inserts all these values into the password table of john_db. Once this is complete, db->insert_id() is called, which returns the id of the object loaded into the database. A DB_STATUS stat object is created. The return id is stored in stat.pwid and all other stat fields are set to default values. The function stat.insert() is called to insert all the status fields into the status table in john_db.

```
class DB_PASSWD : public DB_BASE, public PASSWD{
public:
    DB_PASSWD(DB_CONN* p=0);
    int get_id();
    void db_print(char*);
    void db_parse(MYSQL_ROW &row);
    int check_dup(char*, char*,char*);
    void load_pw (char*,char*,char*,char*,char*,
    char*, char*,char*);
};
```

(5) Class DB_STATUS. Is designed exactly like the password class above, but instead uses a STATUS object as the base and connects to the status table in john_db. Status objects are created only when a password is entered into the database.

```
class DB_STATUS : public DB_BASE, public STATUS{
public:
    DB_STATUS(DB_CONN* p=0);
    int get_id();
    void db_print(char*);
    void db_parse(MYSQL_ROW &row);
    int check_flag(int flg);
    int update_stat_flag_to_working();
    void operator=(STATUS& r) {STATUS::operator=(r);}
};
```

(6) Class DB_BATCH. This class is used just like the other classes above to transfer data between the JtR Database (john_db) and the programs using this data.

```
class DB_BATCH : public DB_BASE, public BATCH{
public:
    DB_BATCH(DB_CONN* p=0);
    int get_id();
    void db_print(char*);
    void db_parse(MYSQL_ROW &row);
    void create_batch(int);
    int update_batch(BATCH &b);
    int load_format(BATCH &b, char **mode);
    int load_cipher(BATCH &b, char **cipher);
    int check_NOT_WORK(char*);
};
```

(7) Class DB_BATCH_SET. This class is used to create a group of batch objects used in the work generation loop as one object.

```
class DB_BATCH_SET:public DB_BASE_SPECIAL{
public:
    DB_BATCH_SET(DB_CONN* p=0);
    BATCH last_batch;
    int items_this_q;

    int load_set(std::vector<BATCH> &b);

};
```

(8) Class DB_PASSWD_ITEM_SET. This class is used to create a group of passwords. This class is used in the Work Generator to retrieve and update passwords.

```
class DB_PASSWD_ITEM_SET: public DB_BASE_SPECIAL{
public:
      DB_PASSWD_ITEM_SET(DB_CONN* p=0);
      PASSWD_ITEM last_item;
      int items_this_q;

      int  enumerate(std::vector<PASSWD_ITEM>&  items,
int btch_id, int stat_flg );
      int update_status(STATUS&);
      int update_passwd(PASSWD &pw);
      int get_passwd(int,char*);
      int update_pw(char*);
};
```

(9) The remainder of the John Database code is located in Appendix A, sections J and K.

D. JOHN CLIENT APPLICATION

The client side development of a BOINC project is focused on processing work units. The Client application operates closer to the normal design of JtR. It is designed to take the starting position and finishing position for a segment of the password key-space the Work Generator creates and returns either a cracked password or a report that no passwords were found. Therefore, the Client application is started by the Core Client application when there is a result unit ready to process and stop when the result unit is processed. That means only one cracking modes will be used every time the client application starts. Since the location in the cracking process is not known to the client, each cracking mode will need to be able to identify where in the key-space to start and stop.

BOINC has an interface between the Core Client and the Client Application on the user's computer. All files and inputs required to run the application have been identified for use with the Client Application. The format of the result units contains the names of the files needed to process each result.

The client application is very similar to the original JtR, because it is removed from John the Ripper database and has very little interaction with the Core Client application on the client's computer. All additions and changes to the program had to be written in C. The work generation program used MySQL databases for data transfer and storage. To minimize the amount of change to the client application, internal databases were used for data transfer and storage.

1. Planning

JtR requires an argument string, and a list of external files to operate. The argument string is required at the time of startup to tell JtR what cracking mode is going to be performed and on what passwords. The external files required are a cracking file, password file, configuration file, word list file, and character set file. The character set, word list and configuration file only change if an update to the program is required. Therefore, they are set to be transferred once to the client's computer when the user first requests to be a part of the project. Any time those files are changed they are automatically transferred to the client computer during the next request for work.

The cracking file and password file are transferred to the client with every result unit. The cracking file contains all the information required for the argument string to start the cracking process. The password file holds the password hashes to be cracked. The basic design is to use the data from the cracking file during the Client application initialization to start JtR in the right cracking mode and at the correct location in the key-space. Then stop JtR at the correct location in the key-space. The original design of the Work Generator made this a fairly strait forward process. The required changes to Incremental cracking mode forced extensive changes to Incremental cracking section of the client JtR, but only minor changes to the rest. The overall plan did not change, but the implementation did.

2. Implementation

The Client application is run in only one cracking mode at a time. Therefore, the batch operation is disabled. The cracking mode to run is retrieved from the crack file. The goal of implementation was to maintain the Client application as close to the original

John the Ripper (JtR) program as possible; firstly to maintain the integrity of the program and secondly to make the implementation simpler. The configuration database structure (III.C.1. b. Configuration Initialization – cfg_init()) was utilized to import and access the crack data file and its data while the program was running. The BOINC Core Client application only required one C++ function to be implemented into the client application to retrieve the filenames of the data files passed to the application. In addition, each of the cracking modes had to be modified to start and stop as required by the data package sent to the client application. Finally, when the work unit was completely processed, a file had to be generated to return the results to the BOINC server complex.

a. Importing Cracking Data

Each result unit generated for a work unit requires a password file and a crack file to complete the check of that work unit. The first problem is relying on the BOINC Core client to run each client application correctly. The Core Client will run the program that is transferred to it without command line inputs. This is problem because the client application uses command line input to control the program. The solution was to bypass the command line input by creating a command line during the initialization section of JtR. The file name of the password file was retrieved from the Core Client with the resolved_filename() function. This function wraps a C++ function call boinc_resolve_filename() which gets the filename of the password or crack file from the Core Client. The crack file (see figure 8) is imported into the configuration database with the configuration initialization function. The crack file has the same format as the John configuration file which allows the data to be imported into different sections of the configuration database for later retrieval.

```
[Crack_Mode]
Mode = -incremental=All
[List.Password_id]
1
2
3
4
5
[Start_Location]
s_loc = 740
s_length = 5
s_fixed = 2
s_count = 11
s_num0 = 8
s_num1 = 8
s_num2 = 11
s_num3 = 1
s_num4 = 4
s_num5 = 0
s_num6 = 0
s_num7 = 0
f_loc = 755
f_length = 2
f_fixed = 0
f_count = 39
f_num0 = 39
f_num1 = 0
f_num2 = 0
f_num3 = 0
f_num4 = 0
f_num5 = 0
s_num6 = 0
s_num7 = 0
```

Figure 8. Example Crack File.

b. Change to db_password

The structure for the struct db_password changed to add a password identification number (pwid) to track the status of passwords throughout the cracking process. The pwid is the identification number used by JtR Database in the BOINC server complex to track an individual password. The db_password field pwid is updated during the password importing phase of the initialization process from the configuration database which contains the cracking data. When the hash is imported into the db_password struct the next pwid number from the [List.Password_id] section of the configuration database is stored in the db_password pwid field. The work generator

loads the passwords in the password file in the same order as the pwids are loaded into the crack file to ensure the pwid is correlated with the correct password.

c. Single Crack Mode

The Client application is designed to run in only one cracking mode at a time. Therefore, the Single Crack mode required no changing because it is so small the Work Generator only generates one work unit. The Single Crack mode will run to completion and terminate the program in normal fashion.

d. Wordlist Crack Mode

The Work Generator is designed to generate work one rule number at a time. The Client application uses the rule number or rule id to identify which rule is to be used to process the wordlist. When the Wordlist Crack mode starts it loads the rule id from the configuration database section defined as the start location from the value s_loc. The Wordlist Crack mode then searches through the rule list until it finds the rule id from the start location. It then verifies it is a valid rule and proceeds to apply that rule to the entire word list. The process goes as follows; the rule is applied to each word, it is converted to a hash, and compared to the list of hashed passwords. If a match is found then the cracked password database (db_cracked) is updated and the remainder of the wordlist is checked against the remaining password hashes.

e. Incremental Crack Mode

The Incremental Crack mode is designed to process only a part of the entire key-space. To accomplish this required significant modification to JtR to enter and exit from the key-space to be searched. Entry is accomplished by using JtR's internal recovery option. The internal recovery option is a part of JtR that allows the user of the program to terminate JtR and restart JtR without having to restart the cracking process. The normal recovery option is operated by storing a recovery file just before the program terminates. When JtR is restarted with the recovery option entered into the command line it will pick up the cracking process where it left off. The Client application takes advantage of this option by retrieving all the normal recovery information from the crack

file stored in the configuration database, when the Client application is started. This allows the client application to jump to the correct place in the key-space to continue searching for passwords.

JtR is designed to continue searching through the key-space from that point. A struct TRACKER was created with all the attributes required to indentify the starting and finishing location of each search through the key-space. In addition, a function end_of_wu() was written to determine if the end of the key-space had been reached. This function is inputted directly following the function call to check a guess for the password. If all the data matches then the program enters the shutdown protocol. As described above the size of each key-space section is the same. However, in most cases that key-space covers multiple sections of the programs search parameters. When a guess matches one of the hashes both the password hash and its plaintext are stored in the cracked password database (db_cracked). The Client application will continue searching the rest of the key-space for the remaining password hashes.

f. Closure of the Client Application

When a cracking mode completes its assigned workunit the Client application generates an output file named as directed by the result unit. This output file contains the pwid, password hash, and plaintext guess for the password hash. If no plaintext is found for the password hash the output file indicates that no password was found. The Client application finishes the shutdown sequence by de-allocating all of memory used for the cracking process and terminates the process. When the Client application terminates the Core Client connects to the BOINC server complex to transmit the output file and the completion of the result unit.

E. VALIDATOR

The validation process required for this project is a bit wise comparator. BOINC gives several example Validators to choose from, a bit wise comparator was chosen to validate all results because the data sets returned are small and no variation of data between two computers should exist. The Validator takes the result units returned from two different clients and checks to ensure they are exactly the same thus giving that

workunit a canonical result. This ensures that a malicious user will not return false information. Should the results not match a third result will be produced and sent to a third user for processing. This process will continue until a canonical result is determined. The criteria for determining a canonical result is established by the operators of the BOINC server complex.

F. ASSIMILATOR

The Assimilator takes the file returned by the client and processes the information storing it in the JtR Database.

1. Planning

BOINC has a fully functional Assimilator already programmed. The only function that is required to be written is process_file(). This functions sole purpose is to take the file returned by the client and store it in the JtR Database.

2. Implementation

The base function that is already written by BOINC is assimilate_handler(). This function verifies the results that have output files ready to process then retrieves the name of that file. That file name is passed as an argument to process_file(). The function process_file() reads the file one line at a time using a function to parse the line. Then it uses the pwid field from the line to search the JtR Database for the correct password. Once the password associated with that pwid is returned, the plaintext of that password hash is updated and stored in the JtR Database. After every line in the file is processed, that result unit is reported complete allowing the next result unit to be processed.

THIS PAGE INTENTIONALLY LEFT BLANK

V. FUTURE DEVELOPMENT

A. NETWORKED HIGH-SPEED PROCESSING

Any new computing device that can be part of the Internet can be utilized by BOINC and therefore this project. For example, the new PS 3s have a processor called the Cell Broadband Engine (Reed, 2008), which is an extremely fast process designed to work with math-intensive applications (i.e., Games). A BOINC Core Client application and Client applications would have to be written to take advantage of this source of computing power, but the capability to utilize this source computing power is endless.

B. VALIDATOR UPDATE

As described above, the Validator takes the results from different computers to verify that a result is not malicious. This process reduces the efficiency of the password cracking process, but is essential to prevent malicious users from affecting the results. The criteria for determining the canonical result is set by the operators of the server complex. The number of correct results, determined by the validation function, can be set to one result if the validation function does not perform a bit wise comparison. The bit wise comparison requires a minimum of two results to establish a canonical result. Since the result files are small the addition of a password checker in the validation function could solve this problem. The checking of all plaintext results would eliminate the need to send out multiple copies of a workunit to determine hostile intent. This checker should be able to catch if the plaintext returned is not the correct password or the default plaintext returned is the password. If the plaintext fails the test the workunit would be regenerated, sent to a different user, and the source of the malicious result would be removed from the users list. Further research should be performed to evaluate the efficiency of this process.

C. REMOVAL OF SINGLE AND WORDLIST CRACKING MODES

More analysis should be performed on the efficiency of password cracking by removing the Single and Wordlist cracking modes. It is assumed that the majority of computer users still use simple word based passwords vice the more complicated and

secure random string of letters, numbers, and special characters. Computer users strengthen word passwords with letter manipulations and substitutions. With the proper dictionary and word manipulation tools, the Wordlist cracking mode could be a very strong supplier of cracked passwords. Testing with a significant number of users is required to validate this assumption and show the usefulness of the Wordlist cracking method. The Single cracking method requires one workunit, therefore it would be difficult to determine the usefulness in the retaining or removing this mode.

D. FULL TESTING ONLINE

To verify the results of this work full testing is still required to determine if it can meet or beat current password cracking techniques. A full operational test would require a dedicated Web site with multiple computers running the server complex and data servers. The number of computers that participate in the project will directly determine the outcome and projections for the test.

VI. SUMMARY

Determining the strength of user passwords or cracking passwords of malicious entities in the world is of great interest to many companies and governments. We demonstrated that it was possible to use the BOINC environment to construct a Distributed John the Ripper. The distributed component of this cracking solution enables multiple platforms and various computing devices the ability to contribute to the password cracking process. Furthermore, having the process work on excess processing power makes the system transparent which a key attribute for any computer user. Significant testing is still required of Distributed John the Ripper to fully test its efficiency and structure. This system was only tested enough to show that it worked, but no data beyond initial testing was completed due to a lack of computer resources to test public source computing.

THIS PAGE INTENTIONALLY LEFT BLANK

LIST OF REFERENCES

Anderson, D. P. (2004). BOINC -A System for Public-Resource Computing and Storage. Paper presented at the Fifth IEEE/ACM International Workshop, Berkeley, CA.

BOINC. (n.d.) Open-source software for volunteer computing and grid computing. Accessed Febuary 1, 2005, from boinc.berkeley.edu

Digital Intelligence. (n.d.). Digital Intelligence FRED SC. Accessed December 1, 2009, from http://www.digitalintelligence.com/products/fredsc/

Distributed.net. (2003). Mission Statement. Accessed November 14, 2006, from www.distributed.net

Drew, J. (n.d.). BOINC Combined Statistics. Accessed May 22, 2008, from boinc.netsoft-online.com/e107_plugins/boinc/bp_home.php

Elcomsoft Co. (n.d.). Elcomsoft Distributed Password Recovery. Accessed December 1, 2009, from http://www.crackpassword.com/products/prs/integpack/edpr/

GECOS. (2009). General Comprehensive Operating System. Wikipedia. Accessed December 2, 2009, from http://en.wikipedia.org/wiki/General_Comprehensive_Operating_System

GIMPS. (2006). The Great Internet Mersenne Prime Search. Accessed May 22, 2008, from www.mersenne.org/prime.htm

Lim, R. (2004). Parallelization of John the Ripper (JtR) using MPI [Report]. University of Nebraska-Lincoln, Computer Science and Engineering.

McDowell, M., Rafail, J., & Hernan, S. (2009). National Cyber Alert System Cyber Security Tip ST04-002. US-Cert United States Computer Emergency Readiness Team. Carnegie Mellon University. Access December 1, 2009, from http://www.us-cert.gov/cas/tips/ST04-002.html

Openwall Project. (n.d.) Openwall Project John the Ripper password cracker. Accessed Febuary 1, 2005, from www.openwall.com/john.

Pippin, A., Hall, B., & Chen, W. (2004). Parallelization of John the Ripper. Report, Santa Barbara:University of California, Santa Barbara.

Reed B. (May 19, 2008). Inside Lockheed Martin's wireless security lab. Accessed August 13, 2008, from http://www.networkworld.com/news/2008/051908-lockheed-martin-wireless-security-lab.html

Roberts, J. L. (August 1, 2005). Keepin' It On the Download [Article], *Newsweek*, p. 42.

Garfinkel, S. & Spafford, G. (n.d.). Chapter 3 Users and Passwords. Practical UNIX & Internet Security . - O'Reilly & Associates, 1999. Accessed December 1, 2009 from http://docstore.mik.ua/orelly/networking/puis/ch03_02.htm

SALT. (2009) Salt (cryptography). Wikipedia. Accessed December 1, 2009, from http://en.wikipedia.org/wiki/Salt_(cryptography)

SOAP. (2007). Simple Object Access Protocol SOAP Version 1.2. World Wide Web Consortium. Accessed May 22, 2008, from www.w3.org/TR/2007/REC-soap12-part0-20070427

TOP500.org. (2007) TOP 500 Supercomputer Sites. Accessed May 22, 2008, from www.top500.org/lists/2007/11

U.S. Census Bureau. (n.d.). U.S. Census Bureau. Accessed May 22, 2008, from www.census.gov

APPENDIX A: WORK GENERATOR CODE

A. JOHN.C

```
/*************************************************************
****
 *  Distributed BOINC Password Cracker- Using John The Ripper
 *
 *  This is a BOINC app that cracks passwords. This part of the
 *  application is the work generator.  All BOINC projects require
 *  some kind of work generator for BOINC to produce result units
 *  to send to clients.
 *
 *  7 June 2005 to use newest BOINC API as of vers. 4.52
 *
 *  John Crumpacker <jrcrumpa@nps.edu>  -  7 June 2005
 *  Department of Computer Science,
 *  Naval Postgraduate School, Monterey, California
 *  @(#) $Version: 1.00$

*********************************************************************
***/
/*
 * This file is part of John the Ripper password cracker,
 * Copyright (c) 1996-2004 by Solar Designer
 * Modified for BOINC by John Crumpacker
 * This is the work generator version of John the Ripper the only
 * purpose is to create work units for client computers.
 */


//Functions added to make this distributed
#include "boinc.h"


// C lib
#include <stdio.h>
#include <unistd.h>
#include <string.h>
#include <stdlib.h>
#include <sys/stat.h>

// John the ripper includes
#include "arch.h"
#include "misc.h"
#include "params.h"
#include "path.h"
#include "memory.h"
#include "list.h"
#include "tty.h"
#include "signals.h"
#include "common.h"
```

```
#include "formats.h"
#include "loader.h"
#include "logger.h"
#include "status.h"
#include "options.h"
#include "john_config.h"
#include "bench.h"
#include "charset.h"
#include "single.h"
#include "wordlist.h"
#include "inc.h"
#include "external.h"

#if CPU_DETECT
extern int CPU_detect(void);
#endif

extern struct fmt_main fmt_DES, fmt_BSDI, fmt_MD5, fmt_BF;
extern struct fmt_main fmt_AFS, fmt_LM;

extern int unshadow(int argc, char **argv);
extern int unafs(int argc, char **argv);
extern int unique(int argc, char **argv);

static struct db_main database;
static struct fmt_main dummy_format;

static void john_register_one(struct fmt_main *format)
{
      if (options.format)
      if (strcmp(options.format, format->params.label)) return;

      fmt_register(format);
}

static void john_register_all(void)
{
      if (options.format) strlwr(options.format);

      john_register_one(&fmt_DES);
      john_register_one(&fmt_BSDI);
      john_register_one(&fmt_MD5);
      john_register_one(&fmt_BF);
      john_register_one(&fmt_AFS);
      john_register_one(&fmt_LM);

      if (!fmt_list) {
            fprintf(stderr,   "Unknown   ciphertext   format   name
requested\n");
            error();
      }
}

static void john_log_format(void)
```

```
{
      int min_chunk, chunk;

      log_event("- Hash type: %.100s (lengths up to %d%s)",
            database.format->params.format_name,
            database.format->params.plaintext_length,
            database.format->methods.split != fmt_default_split ?
            ", longer passwords split" : "");

      log_event("- Algorithm: %.100s",
            database.format->params.algorithm_name);

      chunk       =       min_chunk       =       database.format-
>params.max_keys_per_crypt;
      if (options.flags & (FLG_SINGLE_CHK | FLG_BATCH_CHK) &&
          chunk < SINGLE_HASH_MIN)
                chunk = SINGLE_HASH_MIN;
      if (chunk > 1)
            log_event("- Candidate passwords %s be buffered and "
                  "tried in chunks of %d",
                  min_chunk > 1 ? "will" : "may",
                  chunk);
}

static char *john_loaded_counts(void)
{
      static char s_loaded_counts[80];

      if (database.password_count == 1)
            return "1 password hash";

      sprintf(s_loaded_counts,
            database.salt_count > 1 ?
            "%d password hashes with %d different salts" :
            "%d password hashes with no different salts",
            database.password_count,
            database.salt_count);

      return s_loaded_counts;
}


static void john_load(void)
{
//    struct list_entry *current;
      umask(077);
      fprintf(stderr,"Entering john_load\n");
/*    if (options.flags & FLG_EXTERNAL_CHK)
            ext_init(options.external);

      if (options.flags & FLG_MAKECHR_CHK) {
            options.loader.flags |= DB_CRACKED;
            ldr_init_database(&database, &options.loader);
```

```
            if (options.flags & FLG_PASSWD) {
                  ldr_show_pot_file(&database, POT_NAME);

                  database.options->flags |= DB_PLAINTEXTS;
                  if ((current = options.passwd->head))
                  do {
                        ldr_show_pw_file(&database, current->data);
                  } while ((current = current->next));
            } else {
                  database.options->flags |= DB_PLAINTEXTS;
                  ldr_show_pot_file(&database, POT_NAME);
            }

            return;
      }
//does not need this...
      if (options.flags & FLG_STDOUT) {
            ldr_init_database(&database, &options.loader);
            database.format = &dummy_format;
            memset(&dummy_format, 0, sizeof(dummy_format));
            dummy_format.params.plaintext_length = options.length;
            dummy_format.params.flags = FMT_CASE | FMT_8_BIT;
      }
*/
/*    if (options.flags & FLG_APP) {
            fprintf(stderr,"app_name:%s\n",options.app_name);
            insert_app_name(options.app_name);
      }else{*/
            insert_app_name(DEF_APP_NAME);
/*    }
      if (options.flags & FLG_PASSWD) {
            won't need this... Going to have a different interface
                  to deal with this stuff. Cracked Passwords will be
                  in a MySQL DB*/
/*          if (options.flags & FLG_SHOW_CHK) {
                  options.loader.flags |= DB_CRACKED;
                  ldr_init_database(&database, &options.loader);

                  ldr_show_pot_file(&database, POT_NAME);

                  if ((current = options.passwd->head))
                  do {
                        ldr_show_pw_file(&database, current->data);
                  } while ((current = current->next));

                  printf("%s%d password hash%s cracked, %d left\n",
                        database.guess_count ? "\n" : "",
                        database.guess_count,
                        database.guess_count != 1 ? "es" : "",
                        database.password_count -
                        database.guess_count);

                  return;
            }
```

```
            fprintf(stderr,"Inside john_load\n");
            if (options.flags & (FLG_SINGLE_CHK | FLG_BATCH_CHK))
                  options.loader.flags |= DB_WORDS;
            else
            if (mem_saving_level)
                  options.loader.flags &= ~DB_LOGIN;*/
            ldr_init_database(&database, &options.loader);
/*
            if ((current = options.passwd->head))
            do {
                  ldr_load_pw_file(&database, current->data);
            } while ((current = current->next));

            if ((options.flags & FLG_CRACKING_CHK) &&
                database.password_count) {
                  log_init(LOG_NAME, NULL, options.session);
                  if (status_restored_time)
                        log_event("Continuing      an      interrupted
session");
                  else
                        log_event("Starting a new session");
                  log_event("Loaded      a      total      of      %s",
john_loaded_counts());
            }
//Do not need a pot file... in work gen... maybe in Assimilator...
            ldr_load_pot_file(&database, POT_NAME);
//Do not need to fix the DB... because not stored locally
//          ldr_fix_database(&database);
//will change this to log PASSWD_ITEMS loaded...
            if (database.password_count) {
                  log_event("Remaining %s", john_loaded_counts());
                  printf("Loaded %s (%s [%s])\n",
                        john_loaded_counts(),
                        database.format->params.format_name,
                        database.format->params.algorithm_name);
            } else {
                  log_discard();
                  puts("No password hashes loaded");
            }

            if  ((options.flags  &  FLG_PWD_REQ)  &&  !database.salts)
exit(0);
      }*/
      fprintf(stderr,"exiting john_load\n");
}

static void john_init(int argc, char **argv)
{
#if CPU_DETECT
      if (!CPU_detect()) {
#if CPU_REQ
#if CPU_FALLBACK
#if defined(__DJGPP__) || defined(__CYGWIN32__)
```

```
#error CPU_FALLBACK is incompatible with the current DOS and Win32
code
#endif
            execv(JOHN_SYSTEMWIDE_EXEC    "/"    CPU_FALLBACK_BINARY,
argv);
            perror("execv");
#endif
            fprintf(stderr, "Sorry, %s is required\n", CPU_NAME);
            error();
#endif
      }
#endif
      fprintf(stderr, "Inside John_init\n");
//Should be able to leave this be...
      path_init(argv);

//In params.h JOHN_SYSTEMWIDE is set to 0.. not enabled
#if JOHN_SYSTEMWIDE
      cfg_init(CFG_PRIVATE_FULL_NAME, 1);
      cfg_init(CFG_PRIVATE_ALT_NAME, 1);
#endif
/*Need this for sure... the difference being that it accepts both
john.conf   and   john.ini   files...      Add   template   names   to
john.conf/.ini or
just take commandline. If I modify this can have a default and have
flexiability
for change later on. */
      cfg_init(CFG_FULL_NAME, 1);
      cfg_init(CFG_ALT_NAME, 0);

/*this will create the connection
to the BOINC DB and bring in data from MySQL DB.*/
      boinc_DB_init();



      status_init(NULL, 1);
/* Added Application name to the command line options and removed
all other
 *  functionality.  To  ensure  this  was  only  a  work  generator.  Used
john logging
 * functionality for error reporting.*/
      opt_init(argc, argv);
/*Load all formats*/
      john_register_all();
      common_init();

      sig_init();
//only need parts of this ...
//    john_load();
      fprintf(stderr,"Exit of John_init\n");

}
```

```
static void john_run(void)
{
/*    if (options.flags & FLG_TEST_CHK)
            benchmark_all();
      else
      if (options.flags & FLG_MAKECHR_CHK)
            do_makechars(&database, options.charset);
      else*/
      fprintf(stderr,"Inside John_run\n");
      while ( flag_check() && !daemons_stopped()){
            if (!check_ready_wu()){
            if (process_wu_by_batch(&database))
            {
               fprintf(stderr,"Failed to process batches\n");
               break;
            }
            }else{
                  john_sleep();
            }
      }


            /*if (!(options.flags & FLG_STDOUT)) {
                  status_init(NULL, 1);
                  log_init(LOG_NAME, POT_NAME, options.session);
                  john_log_format();
                  if (cfg_get_bool(SECTION_OPTIONS, NULL, "Idle"))
                        log_event("- Configured to use otherwise idle
"
                              "processor cycles only");
            }*/
            //tty_init();

/*          if (options.flags & FLG_SINGLE_CHK)
                  do_single_crack(&database);
            else
            if (options.flags & FLG_WORDLIST_CHK)
                  do_wordlist_crack(&database, options.wordlist,
                        (options.flags & FLG_RULES) != 0);
            else
            if (options.flags & FLG_INC_CHK)
                  do_incremental_crack(&database, options.charset);
            else
            if (options.flags & FLG_EXTERNAL_CHK)
                  do_external_crack(&database);
            else
            if (options.flags & FLG_BATCH_CHK)
                  do_batch_crack(&database);

            status_print();
            tty_done();
            }
      }*/
}
```

```
static void john_done(void)
{
      path_done();

      if ((options.flags & FLG_CRACKING_CHK) &&
          !(options.flags & FLG_STDOUT)) {
            if (event_abort)
                  log_event("Session aborted");
            else
                  log_event("Session completed");
      }
      log_done();

      check_abort(0);
}

int main(int argc, char **argv)
{
      char *name;

#ifdef __DJGPP__
      if (--argc <= 0) return 1;
      if ((name = strrchr(argv[0], '/')))
            strcpy(name + 1, argv[1]);
      name = argv[1];
      argv[1] = argv[0];
      argv++;
#else
      if (!argv[0])
            name = "";
      else
      if ((name = strrchr(argv[0], '/')))
            name++;
      else
            name = argv[0];
#endif

#ifdef __CYGWIN32__
      if (strlen(name) > 4)
      if (!strcmp(strlwr(name) + strlen(name) - 4, .”exe"))
            name[strlen(name) - 4] = 0;
#endif

      if (!strcmp(name, "unshadow"))
            return unshadow(argc, argv);

      if (!strcmp(name, "unafs"))
            return unafs(argc, argv);

      if (!strcmp(name, "unique"))
            return unique(argc, argv);

      john_init(argc, argv);
```

```
    john_load();
      john_run();
      john_done();

      return 0;
}
```

B. BOINC.H

```
#ifndef _JOHN_BOINC_H
#define _JOHN_BOINC_H

// #include <boinc_db.h>
// #include "john_db.h"

#ifdef __cplusplus
extern "C" {
#endif

/*This function initializes connections to both
the BOINC db and John db*/
void boinc_DB_init();

/*Loads passwords from file into john_db*/
void boinc_pw_load(char *login, char *ciphertext, char *gecos,char
*home,
            char *UID, char *GID, char *shell,char *format);

/*Checks for duplicate passwords (both login and ciphertext match
another password in the list) returns true if no duplicate found.*/
int pw_check_dup(char *login, char *ciphertext, char *format);

int flag_check();

void insert_app_name(char *);

void john_sleep();

int daemons_stopped();

int check_ready_wu();

int process_wu_by_batch(struct db_main *db);

struct BATCH{
    int id;
    int mode_flag;
    int batch_priority;
    int s_loc;//s=start ... Entry or Rule #
    int s_length;
    int s_fixed;
    int s_count;
    int s_num0;
```

```
    int s_num1;
    int s_num2;
    int s_num3;
    int s_num4;
    int s_num5;
    int s_num6;
    int s_num7;
    int f_loc;
    int f_length;
    int f_fixed;
    int f_count;
    int f_num0;
    int f_num1;
    int f_num2;
    int f_num3;
    int f_num4;
    int f_num5;
    int f_num6;
    int f_num7;


};

int gen_work(struct BATCH *b);

#ifdef __cplusplus
}
#endif
#endif
```

C. BOINC.C

```
#include <vector>
#include <algorithm>
#include <stdlib.h>
#include <stdio.h>
#include <string.h>
#include <unistd.h>
//BOINC Backend Libraries
#include "filesys.h"
#include "backend_lib.h"
#include "util.h"
#include "sched_config.h"
#include "sched_util.h"
#include "sched_msgs.h"
#include "parse.h"
#include "crypt.h"
//#include "error_numbers.h"

#include "boinc.h"

//Distributed John Includes
extern "C" {
```

```
#include <sys/stat.h>
#include "misc.h"
#include "params.h"
#include "john_config.h"
#include "logger.h"
#include "single.h"
#include "wordlist.h"
#include "inc.h"
#include "formats.h"
}

#include <boinc_db.h>
#include "john_db.h"

DB_APP app;
//R_RSA_PRIVATE_KEY key;
SCHED_CONFIG conf;



int num_ready_wu;
int wu_count = 0;
int trigger = 0;


void boinc_DB_init(){
        int retval;
        int password_count = 0;
      char *DBname;
      char *Host_name;
      char *User_name;
      char *pwd;

        retval = conf.parse_file(.”.");
        if (retval){
                fprintf(stderr, "Can't parse BOINC config file\n");
                error();
        }
        retval    =    boinc_db.open(conf.db_name,    conf.db_host,
conf.db_user, conf.db_passwd);
        if (retval){
            fprintf(stderr,"Can not open BOINC db\n");
                error();
        }
      DBname = cfg_get_param(SECTION_BOINC,NULL,"DB_name");

      if (!DBname)
            DBname = DB_NAME;
      //need to add user to params... will not have a password would
have to
      //figure another way to save the password safely... Don't want
to deal
      //with that.
      Host_name = cfg_get_param(SECTION_BOINC,NULL,"Host_name");
```

```
      User_name = cfg_get_param(SECTION_BOINC,NULL,"User_name");
      pwd =cfg_get_param(SECTION_BOINC,NULL,"Pass_word");

//    fprintf(stderr,"DB_name    %s,    Host_name    %s,    User_name
%s\n",DBname,Host_name,User_name);

      retval = john_db.open(DBname,Host_name,User_name,pwd);
      if (retval){
            fprintf(stderr,"Can not open John DB\n");
            error();
      }
//    fprintf(stderr,"exit of boinc_DB_init\n");
      num_ready_wu = 0;
      num_ready_wu = cfg_get_int(SECTION_BOINC,NULL,"Max_wu");

      if (!(num_ready_wu) ){
            fprintf(stderr,"Failed to get Number of ready"
                        " workunits\n");
            error();
      }
}//END boinc_DB_init();

/*this function is not in John_db because it is a query from the
 *boinc database... The purpose of this function is to count the
number
 *workunits created to the number of results that have been
completed or have  *been sent to be processed. This makes sure all
the work generated has been
 *processed before the work generator will generate more work.
 */
int check_ready_wu(){
      DB_WORKUNIT wu;
      DB_RESULT re;
    char query_wu[MAX_QUERY_LEN];
    char query_res[MAX_QUERY_LEN];
      int retval;
      int total_wu_processed = 0;//returned complete or sent
      int total_wu_count = 0;
      MYSQL_ROW row;
      MYSQL_RES *res;
      sprintf(query_wu,"select count(*) from workunit");

      retval = wu.db->do_query(query_wu);
      if (retval) return mysql_errno(wu.db->mysql);
      res= mysql_store_result(wu.db->mysql);
      if (!res) return mysql_errno(wu.db->mysql);
      row = mysql_fetch_row(res);
      if (!row){
            mysql_free_result(res);
            fprintf(stderr,"Error counting workunits\n");
                return 1;
      }
      total_wu_count = atoi(row[0]);
      mysql_free_result(res);
```

```
    sprintf(query_res,"select count(*) from result where "
            "validate_state != '%d'"
            ,VALIDATE_STATE_INIT);
    retval = re.db->do_query(query_res);

      if (retval) return mysql_errno(wu.db->mysql);
      res= mysql_store_result(wu.db->mysql);
      if (!res) return mysql_errno(wu.db->mysql);
      row = mysql_fetch_row(res);
      if (!row){
          mysql_free_result(res);
        fprintf(stderr,"Failed to count processed results\n");
        return 1;
    }
    total_wu_processed = atoi(row[0]);
    mysql_free_result(res);

      if ( total_wu_count <= total_wu_processed ){
            return 0;
      }
      return 1;
}



void john_sleep(){
      int retval;
      retval = sleep(6);
      if (retval){
            fprintf(stderr,"Sleep   interupted   by   amount:   %d\n",
retval);
      }
}

int daemons_stopped(){
      check_stop_daemons();
      return 0;
}

void insert_app_name(char *app_name){
      char buf[256];
      int retval;
      strcpy(app.name,app_name);
      sprintf(buf,"where name='%s'",app.name);
      retval = app.lookup(buf);
      if (retval){
            fprintf(stderr,"Failed      to      find      application
%s\n",app.name);
            error();
      }
}
```

```
void boinc_pw_load(char *login, char *ciphertext, char *gecos, char
*home,
            char *UID, char *GID, char *shell, char *format){
      DB_PASSWD pw;
      pw.clear();
      pw.load_pw(login,ciphertext,gecos,home,UID,GID,shell,format);

}//end boinc_load_pw

/*Looks at the status table. Returns true if it finds an instantance
of the flag it is looking for.*/
int flag_check(){
      DB_STATUS stat;
      stat.clear();

      if(stat.check_flag(NOT_WORKED)){
        if(stat.update_stat_flag_to_working()){
            fprintf(stderr,"Failed   to   update   status_flag   to
working");
            return 0;
        }
        return 1;
    }
      else if(stat.check_flag(WORKING))
          return 1;


      return 0;
}//end flag_check

void batch_load_format(BATCH &b, char **mode){
    DB_BATCH con;
    if (con.load_format(b, mode)){
        fprintf(stderr,"FAILED to load format\n");
    }
}

void batch_load_cipher(struct BATCH *b, char **cipher){
    DB_BATCH con;
    if (con.load_cipher(*b,cipher)){
        fprintf(stderr,"Failed to load cipher\n");
    }
}

void      reload_b(std::vector<BATCH>      &target,std::vector<BATCH>
&source){
    while(!source.empty()){
        BATCH *new_b= new BATCH;
        batch_equal(new_b,source.back());
        source.pop_back();
        target.push_back(*new_b);
    }
}
void clean_batch_set(std::vector<BATCH> &b){
```

```
    std::vector<BATCH> temp;
    std::vector<BATCH> clean;

    int s_or_w_pres = 0;   //single or wordlist mode present
    int start_inc =0;
    while(!b.empty()){
        BATCH *b_tmp;
        b_tmp=new BATCH;
        batch_equal(b_tmp,b.back());
        b.pop_back();
        if      ((b_tmp->mode_flag==SINGLE_MODE      ||      b_tmp-
>mode_flag==WORDLIST_MODE)
                && !start_inc ){
            temp.push_back(*b_tmp);
            s_or_w_pres=1;
        }else     if     (b_tmp->mode_flag==INCREMENTAL_MODE     &&
!s_or_w_pres){
            temp.push_back(*b_tmp);
            start_inc=1;
        }else    if    (b_tmp->mode_flag==SINGLE_MODE    ||    b_tmp-
>mode_flag==WORDLIST_MODE){
            clean.push_back(*b_tmp);
        }
    }
    /*if clean has anything it will be single or wordlist batches
     *and the first batch was an incremental batch. if the first
batch
     *was a single or wordlist temp will only contain those. if only
     *incremental then temp is all incremental.
     */
    if (!clean.empty())
        reload_b(b,clean);
    else
        reload_b(b,temp);

}
void load_jtr_db_formats(struct db_main *db, BATCH &b){
    char *format;

    batch_load_format(b,&format);

   if ((db->format = fmt_list))
   {
        do{
            if(!strcmp(format,db->format->params.format_name)){
/*                fprintf(stderr,"Inside form loop length= %d\n"
                      " Format_name=%s\n"
                      ,db->format->params.plaintext_length
                      ,db->format->params.format_name);
*/
                      fmt_init(db->format);
                break;
            }
        } while ((db->format = db->format->next));
```

```
   }


}

int process_wu_by_batch(struct db_main *db){
    DB_BATCH_SET b_con_set;//batch set controller
    DB_BATCH   b_con;// batch controller
    int local_wu_count = 0;
    std::vector<BATCH> batch_set;

    if(b_con_set.load_set(batch_set)){
        fprintf(stderr,"Failed to load batch_set\n");
        return 1;
    }
    clean_batch_set(batch_set);

    while(!batch_set.empty() && !daemons_stopped()){
        BATCH *new_batch;
        new_batch = new BATCH;

        batch_equal(new_batch,batch_set.back());
        batch_set.pop_back();
        while (local_wu_count < new_batch->batch_priority
                                        && !daemons_stopped()){
            load_jtr_db_formats(db,*new_batch);
            if (new_batch->mode_flag==SINGLE_MODE){
                do_single_crack(db,new_batch);
                if (b_con.update_batch(*new_batch)){
                    fprintf(stderr,"Failed   to   update   batch   for
single\n");
                    break;
                }
            }else if (new_batch->mode_flag==WORDLIST_MODE){
                if (!db->format)
                    fprintf(stderr,"NO db->format loaded\n");
                else
                    fprintf(stderr,"db->Format->length = %d\n"
                              ,db->format->params.plaintext_length);
                do_wordlist_crack(db,new_batch,1);
                if (b_con.update_batch(*new_batch)){
                    fprintf(stderr,"Failed   to   update   batch   for
wordlist\n");
                    break;
                }
            }else if (new_batch->mode_flag==INCREMENTAL_MODE){
                do_incremental_crack(db,new_batch);
                if (b_con.update_batch(*new_batch)){
                    fprintf(stderr,"Failed   to   update   batch   for
incremental\n");
                    break;
                }
            }
            local_wu_count++;
```

```
        }

    }

    return 0;


}


int pw_check_dup(char *login,char *password_hash,char *format){
      DB_PASSWD pw;
      pw.clear();
      return pw.check_dup(login,password_hash,format);
}//end pw_check_dup
void  reload(std::vector<PASSWD_ITEM>  &t,  std::vector<PASSWD_ITEM>
&s){
    while (!s.empty()){
        PASSWD_ITEM i;
        i.clear();
        i=s.back();
        t.push_back(i);
        s.pop_back();
    }
}

/* Creates a password file to send to a client.*/
int create_pw_file(std::vector<PASSWD_ITEM> &items, FILE* f){
//    fprintf(stderr,"in create_pw_file\n");
    std::vector<PASSWD_ITEM> temp;
    for(int i=0;i<items.size();i++){
/*                          fprintf(stderr,"%s:%s:%d:%d:%s:%s:%s\n",
items[i].pw.login,
                items[i].pw.passwd_hash, items[i].pw.UID,
                items[i].pw.GID, items[i].pw.GECOS,
                items[i].pw.home_dir,items[i].pw.shell);
*/
        fprintf(f,"%s:%s:%d:%d:%s:%s:%s\n", items[i].pw.login,
                items[i].pw.passwd_hash, items[i].pw.UID,
                items[i].pw.GID, items[i].pw.GECOS,
                items[i].pw.home_dir,items[i].pw.shell);


    }

//    fprintf(stderr,"create_pw_file exit\n");
      return 0;
}

/*  The  crack  file  will  instruct  the  client  how  to  conduct  the
cracking
 * of the password file sent to it. The client will treat the crack
file
```

```
 * like a configuration file that way all fields will be easily
retreivable*/
int create_cr_file(std::vector<PASSWD_ITEM>& items, FILE* f, BATCH&
b){
//    fprintf(stderr,"in create_cr_file\n");
      fprintf(f,"[Crack_Mode]\n");
      switch (b.mode_flag) {

      case 0 : log_event("Incorrect Flag set create_cr_file");
             return 1;
      case 1 : fprintf(f,"Mode = -single\n");
             break;
      case 2 : fprintf(f,"Mode = -wordlist\n");
             fprintf(f,"Req_Mode = -rules\n");
             break;
      case 3 : fprintf(f,"Mode = -incremental=All\n");
             break;
           return 1;
      }
      fprintf(f,"[List.Password_id]\n");
      for (int i=0;i<items.size();i++){
           fprintf(f,"%d\n",items[i].pw.id);
      }
      fprintf(f,"[Start_Location]\n");
    if (b.mode_flag==SINGLE_MODE || b.mode_flag == WORDLIST_MODE){
        fprintf(f,"s_loc = %d\n",b.s_loc);
    }else if (b.mode_flag==INCREMENTAL_MODE){
        fprintf(f,"s_loc = %d\n",b.s_loc);
        fprintf(f,"s_length = %d\n",b.s_length);
        fprintf(f,"s_fixed = %d\n",b.s_fixed);
        fprintf(f,"s_count = %d\n",b.s_count);
        fprintf(f,"s_num0 = %d\n",b.s_num0);
        fprintf(f,"s_num1 = %d\n",b.s_num1);
        fprintf(f,"s_num2 = %d\n",b.s_num2);
        fprintf(f,"s_num3 = %d\n",b.s_num3);
        fprintf(f,"s_num4 = %d\n",b.s_num4);
        fprintf(f,"s_num5 = %d\n",b.s_num5);
        fprintf(f,"s_num6 = %d\n",b.s_num6);
        fprintf(f,"s_num7 = %d\n",b.s_num7);
        fprintf(f,"f_loc = %d\n",b.f_loc);
        fprintf(f,"f_length = %d\n",b.f_length);
        fprintf(f,"f_fixed = %d\n",b.f_fixed);
        fprintf(f,"f_count = %d\n",b.f_count);
        fprintf(f,"f_num0 = %d\n",b.f_num0);
        fprintf(f,"f_num1 = %d\n",b.f_num1);
        fprintf(f,"f_num2 = %d\n",b.f_num2);
        fprintf(f,"f_num3 = %d\n",b.f_num3);
        fprintf(f,"f_num4 = %d\n",b.f_num4);
        fprintf(f,"f_num5 = %d\n",b.f_num5);
        fprintf(f,"f_num6 = %d\n",b.f_num6);
        fprintf(f,"f_num7 = %d\n",b.f_num7);

      }
```

```
//    fprintf(stderr,"create_cr_file exit\n");
      return 0;
}
int gen_files(std::vector<PASSWD_ITEM> &items,BATCH &b){
      int retval;
      FILE *pwf;
      FILE *crf;
      char pw_file_name[256];
      char cr_file_name[256];
      char **input_files = new char*[5];
      char wu_name[256];
      char path_pw[256];
      char path_cr[256];
      char wu_temp_path[256];
      char wu_template[65536];
      int count = 0;
    std::vector<PASSWD_ITEM> temp;

    for (int i=0;i<5;i++)
            input_files[i]= new char[256];
      sprintf(input_files[count++],"john_boinc.conf");
      sprintf(input_files[count++],"all.chr");
      sprintf(input_files[count++],"password.lst");
//    fprintf(stderr,"In work loop\n");
      sprintf(pw_file_name,"pw_%d_%d_%d_%d%d%d%d%d%d%d%d.pw"
            ,b.id,b.mode_flag,b.s_loc,b.s_num0,b.s_num1,b.s_num2
            ,b.s_num3,b.s_num4,b.s_num5,b.s_num6,b.s_num7);
      retval = dir_hier_path(pw_file_name,conf.download_dir

,conf.uldl_dir_fanout,true,path_pw,true);
//    fprintf(stderr,"path_pw:%s\n",path_pw);
      if (retval)
            fprintf(stderr,"Failed     to     load     path     for
%s\n",pw_file_name);

      strncpy(input_files[count++],pw_file_name,sizeof(pw_file_name)
);
      pwf = fopen(path_pw, "w");
      if (!pwf){
            fprintf(stderr,"File not found %s\n",pw_file_name);
            error();
      }

      if (create_pw_file(items,pwf))
            fprintf(stderr,"Could not create pw file\n");
      fclose(pwf);

      sprintf(cr_file_name,"cr_%d_%d_%d_%d%d%d%d%d%d%d%d.cr",b.id,b.
mode_flag

,b.s_loc,b.s_num0,b.s_num1,b.s_num2,b.s_num3,b.s_num4,b.s_num5
            ,b.s_num6,b.s_num7);
      retval = dir_hier_path(cr_file_name,conf.download_dir
```

```
,conf.uldl_dir_fanout,true,path_cr,true);
      if (retval)
            fprintf(stderr,"Failed     to     load     path     for
%s\n",cr_file_name);
      strcpy(input_files[count++],cr_file_name);
      crf = fopen(path_cr, "w");
      if (!crf){
            fprintf(stderr,"Could not create cr file\n");
            error();
      }

    create_cr_file(items,crf,b);
      fclose(crf);
      DB_WORKUNIT wu;
      sprintf(wu.name,"pw_%d_%d_%d_%d%d%d%d%d%d%d%d"
            ,b.id,b.mode_flag,b.s_loc,b.s_num0,b.s_num1,b.s_num2
            ,b.s_num3,b.s_num4,b.s_num5,b.s_num6,b.s_num7);

      wu.appid=app.get_id();
      strcpy(wu.app_name,app.name);
/*    fprintf(stderr,"wu.name:%s\n"
            "wu.appid:%d\n"
                  "wu_temp:%s\n"
                  "re_temp:%s\n"
                  "conf.temp:%s\n",wu.name,wu.appid,
                  WU_TEMPLATE,RE_TEMPLATE,
                  conf.template_dir);
*/    sprintf(wu_temp_path,"%s%s",conf.template_dir,WU_TEMPLATE);
//    fprintf(stderr,"before                              read_filename
call:%s,%d\n",wu_temp_path,conf.uldl_dir_fanout);
      retval=
read_filename(wu_temp_path,wu_template,sizeof(wu_template));
//    fprintf(stderr,"after                               read_filename
call:%s,%s,%d\n",wu_temp_path,wu_template,sizeof(wu_template));
      if (retval) {
            fprintf(stderr,"Failed  to  read  workunit  template  file
'%s'\n"
                ,wu_temp_path);
            error();
      }
/*    fprintf(stderr,"before create work\n"
                  "name:%s appid:%s\n"
                  "inputfilenames:         %s,         %s,         \n
,%s,%s,%s\n",wu.name,wu.app_name,
                  input_files[0]   ,input_files[1],   input_files[2],
input_files[3], input_files[4]);
      fprintf(stderr,"template
file%s%s",conf.template_dir,WU_TEMPLATE);
*/
    retval=      create_work(wu,      wu_template,      RE_TEMPLATE,
conf.template_dir, (const char**)input_files, count, conf);
//    fprintf(stderr,"after create work\n");
      if (retval) {
```

```
            fprintf(stderr,"Work Not Created\n");
            return 1;
      }
    return 0;


}
void update_wu_generated(std::vector<PASSWD_ITEM> &i){
    std::vector<PASSWD_ITEM> temp;
    DB_PASSWD_ITEM_SET control;

    while(!i.empty()){
        PASSWD_ITEM new_item;
        new_item.clear();
        new_item=i.back();
        i.pop_back();
        new_item.pw_stat.workunits_gen++;
        control.update_status(new_item.pw_stat);
        temp.push_back(new_item);
    }
    while(!temp.empty()){
        PASSWD_ITEM new_item;
        new_item.clear();
        new_item=temp.back();
        temp.pop_back();
        i.push_back(new_item);
    }
}

/* This function is called from all the cracking mode files in John
 * flag = Crack Mode being performed, num_rules is the total number
of rules
 * to completely process each password. gen_work calls create_work
 * which creates workunits and results from templates passed in.
 *  NOTE:  I  added  template_dir  to  the  BOINC  config  file,  that
includes all
 * the functionality required to support it.
 */
int gen_work(struct BATCH *b){
      DB_BATCH batch;
      std::vector<PASSWD_ITEM> items;
    std::vector<PASSWD_ITEM> temp;
      DB_PASSWD_ITEM_SET pw_set;
      umask(0222);
      int retval;
      pw_set.enumerate(items,b->id,WORKING);
      if (items.empty()){
        fprintf(stderr,"Failed to enumerate pw_set\n");
        return 1;
    }
    reload(temp,items);
    reload(items,temp);
    if (gen_files(items,*b)){
        return 1;
```

```
    }

    update_wu_generated(items);

    return 0;
}
```

D. CRACKER.H

```
/*
 * This file is part of John the Ripper password cracker,
 * Copyright (c) 1996-99 by Solar Designer
 */

/*
 * Cracking routines.
 */

#ifndef _JOHN_CRACKER_H
#define _JOHN_CRACKER_H

#include "loader.h"

/*
 * Initializes the cracker for a password database (should not be
empty).
 * If fix_state() is not NULL, it will be called when key buffer
becomes
 * empty, its purpose is to save current state for possible recovery
in
 * the future. If guesses is not NULL, the cracker will save guessed
keys
 * in there (the caller must make sure there's room).
 */
extern void crk_init(struct db_main *db, void (*fix_state)(void),
      struct db_keys *guesses);

/*
 * Tries the key against all passwords in the database (should not
be empty).
 * The return value is non-zero if aborted or everything got cracked
(event
 * flags can be used to find out which of these has happened).
 */
extern int crk_process_key(char *key);

/*
 * Resets the guessed keys buffer and processes all the buffered
keys for
 * this salt. The return value is the same as for crk_process_key().
 */
extern int crk_process_salt(struct db_salt *salt);
```

```
/*
 * Return current keys range, crk_get_key2() may return NULL if
there's only
 * one key. Note: these functions may share a static result buffer.
 */
extern char *crk_get_key1(void);
extern char *crk_get_key2(void);

/*
 * Processes all the buffered keys (unless aborted).
 */
extern void crk_done(void);

#endif
```

E. CRACKER.C

```
/*
 * This file is part of John the Ripper password cracker,
 * Copyright (c) 1996-2003 by Solar Designer
 */

#include <string.h>

#include "arch.h"
#include "misc.h"
#include "math.h"
#include "params.h"
#include "memory.h"
#include "signals.h"
#include "idle.h"
#include "formats.h"
#include "loader.h"
#include "logger.h"
#include "status.h"
#include "recovery.h"

#ifdef index
#undef index
#endif

static struct db_main *crk_db;
static struct fmt_params crk_params;
static struct fmt_methods crk_methods;
static int crk_key_index, crk_last_key;
static void *crk_last_salt;
static void (*crk_fix_state)(void);
static struct db_keys *crk_guesses;
static int64 *crk_timestamps;
static char crk_stdout_key[PLAINTEXT_BUFFER_SIZE];

static void crk_dummy_set_salt(void *salt)
{
```

```
}

static void crk_dummy_fix_state(void)
{
}

static void crk_init_salt(void)
{
	if (!crk_db->salts->next) {
		crk_methods.set_salt(crk_db->salts->salt);
		crk_methods.set_salt = crk_dummy_set_salt;
	}
}

void crk_init(struct db_main *db, void (*fix_state)(void),
	struct db_keys *guesses)
{
	char *where;
	size_t size;

	if (db->loaded)
	if ((where = fmt_self_test(db->format))) {
		log_event("! Self test failed (%s)", where);
		fprintf(stderr, "Self test failed (%s)\n", where);
		error();
	}

	crk_db = db;
	memcpy(&crk_params,	&db->format->params,	sizeof(struct
fmt_params));
	memcpy(&crk_methods,	&db->format->methods,	sizeof(struct
fmt_methods));

	if (db->loaded) crk_init_salt();
	crk_last_key = crk_key_index = 0;
	crk_last_salt = NULL;

	if (fix_state)
		(crk_fix_state = fix_state)();
	else
		crk_fix_state = crk_dummy_fix_state;

	crk_guesses = guesses;

	if (db->loaded) {
		size = crk_params.max_keys_per_crypt * sizeof(int64);
		memset(crk_timestamps = mem_alloc(size), -1, size);
	} else
		crk_stdout_key[0] = 0;

	rec_save();

	idle_init();
}
```

```
static int crk_process_guess(struct db_salt *salt, struct
db_password *pw,
	int index)
{
	int dupe;
	char *key;
	struct db_salt *search_salt;
	struct db_password *search_pw;

	dupe = !memcmp(&crk_timestamps[index], &status.crypts,
sizeof(int64));
	crk_timestamps[index] = status.crypts;

	key = crk_methods.get_key(index);

	log_guess(crk_db->options->flags & DB_LOGIN ? pw->login : "?",
		dupe ? NULL : pw->source, key);

	crk_db->password_count--;
	crk_db->guess_count++;
	status.guess_count++;

	if (crk_guesses && !dupe) {
		strnfcpy(crk_guesses->ptr, key,
crk_params.plaintext_length);
		crk_guesses->ptr += crk_params.plaintext_length;
		crk_guesses->count++;
	}

	if (pw == salt->list) {
		salt->list = pw->next;

		ldr_update_salt(crk_db, salt);

		if (!salt->list) {
			crk_db->salt_count--;

			if (salt == crk_db->salts) {
				crk_db->salts = salt->next;
			} else {
				search_salt = crk_db->salts;
				while (search_salt->next != salt)
					search_salt = search_salt->next;
				search_salt->next = salt->next;
			}

			if (crk_db->salts) crk_init_salt(); else return 1;
		}
	} else {
		search_pw = salt->list;
		while (search_pw->next != pw)
			search_pw = search_pw->next;
		search_pw->next = pw->next;
```

```
            ldr_update_salt(crk_db, salt);
      }

      return 0;
}

static int crk_process_event(void)
{
      event_pending = 0;

      if (event_save) {
            event_save = 0;
            rec_save();
      }

      if (event_status) {
            event_status = 0;
            status_print();
      }

      return event_abort;
}

static int crk_password_loop(struct db_salt *salt)
{
      struct db_password *pw;
      int index;

#if !OS_TIMER
      sig_timer_emu_tick();
#endif

      idle_yield();

      if (event_pending)
      if (crk_process_event()) return 1;

      crk_methods.crypt_all(crk_key_index);

      status_update_crypts(salt->count * crk_key_index);

      if (salt->hash_size < 0) {
            pw = salt->list;
            do {
                  if (crk_methods.cmp_all(pw->binary, crk_key_index))
                  for (index = 0; index < crk_key_index; index++)
                  if (crk_methods.cmp_one(pw->binary, index))
                  if (crk_methods.cmp_exact(pw->source, index)) {
                        if (crk_process_guess(salt, pw, index))
                              return 1;
                        else
                              break;
                  }
```

```
            } while ((pw = pw->next));
      } else
      for (index = 0; index < crk_key_index; index++) {
            if ((pw = salt->hash[salt->index(index)]))
            do {
                  if (crk_methods.cmp_one(pw->binary, index))
                  if (crk_methods.cmp_exact(pw->source, index))
                  if (crk_process_guess(salt, pw, index))
                        return 1;
            } while ((pw = pw->next_hash));
      }

      return 0;
}

static int crk_salt_loop(void)
{
      struct db_salt *salt;

      salt = crk_db->salts;
      do {
            crk_methods.set_salt(salt->salt);
            if (crk_password_loop(salt)) return 1;
      } while ((salt = salt->next));

      crk_last_key = crk_key_index; crk_key_index = 0;
      crk_last_salt = NULL;
      crk_fix_state();

      crk_methods.clear_keys();

      return 0;
}

int crk_process_key(char *key)
{
      if (crk_db->loaded) {
            crk_methods.set_key(key, crk_key_index++);

            if (crk_key_index >= crk_params.max_keys_per_crypt)
                  return crk_salt_loop();

            return 0;
      }

#if !OS_TIMER
      sig_timer_emu_tick();
#endif

      if (event_pending)
      if (crk_process_event()) return 1;

      puts(strnzcpy(crk_stdout_key, key, crk_params.plaintext_length
+ 1));
```

```
	status_update_crypts(1);
	crk_fix_state();

	return 0;
}

int crk_process_salt(struct db_salt *salt)
{
	char *ptr;
	char key[PLAINTEXT_BUFFER_SIZE];
	int count, index;

	if (crk_guesses) {
		crk_guesses->count = 0;
		crk_guesses->ptr = crk_guesses->buffer;
	}

	if (crk_last_salt != salt->salt)
		crk_methods.set_salt(crk_last_salt = salt->salt);

	ptr = salt->keys->buffer;
	count = salt->keys->count;
	index = 0;

	crk_methods.clear_keys();

	while (count--) {
		strnzcpy(key, ptr, crk_params.plaintext_length + 1);
		ptr += crk_params.plaintext_length;

		crk_methods.set_key(key, index++);
		if (index >= crk_params.max_keys_per_crypt || !count) {
			crk_key_index = index;
			if (crk_password_loop(salt)) return 1;
			if (!salt->list) return 0;
			index = 0;
		}
	}

	return 0;
}

char *crk_get_key1(void)
{
	if (crk_db->loaded)
		return crk_methods.get_key(0);
	else
		return crk_stdout_key;
}

char *crk_get_key2(void)
{
	if (crk_key_index > 1)
```

```
            return crk_methods.get_key(crk_key_index - 1);
      else
      if (crk_last_key > 1)
            return crk_methods.get_key(crk_last_key - 1);
      else
            return NULL;
}

void crk_done(void)
{
      if (crk_db->loaded) {
            if (crk_key_index && crk_db->salts && !event_abort)
                  crk_salt_loop();

            MEM_FREE(crk_timestamps);
      }
}
```

F. INC.H

```
/*
 * This file is part of John the Ripper password cracker,
 * Copyright (c) 1996-98 by Solar Designer
 */

/*
 * Incremental mode cracker.
 */

#ifndef _JOHN_INC_H
#define _JOHN_INC_H

#include "loader.h"
//#include "boinc.h"
/*
 * Runs the incremental mode cracker.
 */
extern void do_incremental_crack(struct db_main *db, struct BATCH
*b);

#endif
```

G. INC.C

```
/*
 * This file is part of John the Ripper password cracker,
 * Copyright (c) 1996-2004 by Solar Designer
 */

#include <stdio.h>
#include <string.h>
```

```
#include "arch.h"
#include "misc.h"
#include "params.h"
#include "path.h"
#include "memory.h"
#include "signals.h"
#include "formats.h"
#include "loader.h"
#include "logger.h"
#include "status.h"
#include "recovery.h"
#include "john_config.h"
#include "charset.h"
#include "external.h"
#include "cracker.h"
#include "boinc.h"

extern struct fmt_main fmt_LM;

typedef char (*char2_table)
      [CHARSET_SIZE + 1][CHARSET_SIZE + 1];
typedef char (*chars_table)
      [CHARSET_SIZE + 1][CHARSET_SIZE + 1][CHARSET_SIZE + 1];

static int rec_compat;
static int rec_entry;
static int rec_numbers[CHARSET_LENGTH];
static int current_wu_size;

static int entry;
static int numbers[CHARSET_LENGTH];

static void save_state(FILE *file)
{
      int pos;

      fprintf(file,     "%d\n%d\n%d\n",     rec_entry,     rec_compat,
CHARSET_LENGTH);
      for (pos = 0; pos < CHARSET_LENGTH; pos++)
            fprintf(file, "%d\n", rec_numbers[pos]);
}

static int restore_state(FILE *file)
{
      int length;
      int pos;

      if (fscanf(file, "%d\n", &rec_entry) != 1) return 1;
      rec_compat = 1;
      length = CHARSET_LENGTH;
      if (rec_version >= 2) {
            if (fscanf(file, "%d\n%d\n", &rec_compat, &length) != 2)
                  return 1;
            if ((unsigned int)rec_compat > 1) return 1;
```

```
            if ((unsigned int)length > CHARSET_LENGTH) return 1;
      }
      for (pos = 0; pos < length; pos++) {
            if (fscanf(file, "%d\n", &rec_numbers[pos]) != 1) return
1;
            if  ((unsigned  int)rec_numbers[pos]  >=  CHARSET_SIZE)
return 1;
      }

      return 0;
}

static void fix_state(void)
{
      rec_entry = entry;
      memcpy(rec_numbers, numbers, sizeof(rec_numbers));
}

static void inc_format_error(char *charset)
{
      log_event("! Incorrect charset file format: %.100s", charset);
      fprintf(stderr,  "Incorrect  charset  file  format:  %s\n",
charset);
      error();
}

static void inc_new_length(unsigned int length,
      struct charset_header *header, FILE *file, char *charset,
      char *char1, char2_table char2, chars_table *chars)
{
      long offset;
      int value, pos, i, j;
      char *buffer;
      int count;

      log_event("- Switching to length %d", length + 1);

      char1[0] = 0;
      if (length)
            memset(char2, 0, sizeof(*char2));
      for (pos = 0; pos <= (int)length - 2; pos++)
            memset(chars[pos], 0, sizeof(**chars));

      offset =
            (long)header->offsets[length][0] +
            ((long)header->offsets[length][1] << 8) +
            ((long)header->offsets[length][2] << 16) +
            ((long)header->offsets[length][3] << 24);
      if (fseek(file, offset, SEEK_SET)) pexit("fseek");

      i = j = pos = -1;
      if ((value = getc(file)) != EOF)
      do {
            if (value != CHARSET_ESC) {
```

```
            switch (pos) {
            case -1:
                  inc_format_error(charset);

            case 0:
                  buffer = char1;
                  break;

            case 1:
                  if (j < 0)
                        inc_format_error(charset);
                  buffer = (*char2)[j];
                  break;

            default:
                  if (i < 0 || j < 0)
                        inc_format_error(charset);
                  buffer = (*chars[pos - 2])[i][j];
            }

            buffer[count = 0] = value;
            while ((value = getc(file)) != EOF) {
                  buffer[++count] = value;
                  if (value == CHARSET_ESC) break;
                  if (count >= CHARSET_SIZE)
                        inc_format_error(charset);
            }
            buffer[count] = 0;

            continue;
      }

      if ((value = getc(file)) == EOF) break; else
      if (value == CHARSET_NEW) {
            if ((value = getc(file)) != (int)length) break;
            if ((value = getc(file)) == EOF) break;
            if ((unsigned int)value > length)
                  inc_format_error(charset);
            pos = value;
      } else
      if (value == CHARSET_LINE) {
            if (pos < 0)
                  inc_format_error(charset);
            if ((value = getc(file)) == EOF) break;
            if ((unsigned int)(i = value) > CHARSET_SIZE)
                  inc_format_error(charset);
            if ((value = getc(file)) == EOF) break;
            if ((unsigned int)(j = value) > CHARSET_SIZE)
                  inc_format_error(charset);
      } else
            inc_format_error(charset);

      value = getc(file);
} while (value != EOF);
```

```
      if (value == EOF) {
            if (ferror(file))
                  pexit("getc");
            else
                  inc_format_error(charset);
      }
}

static void expand(char *dst, char *src, int size)
{
      char *dptr = dst, *sptr = src;
      int count = size;
      char present[CHARSET_SIZE];

      memset(present, 0, sizeof(present));
      while (*dptr) {
            if (--count <= 1) return;
            present[ARCH_INDEX(*dptr++) - CHARSET_MIN] = 1;
      }

      while (*sptr)
      if (!present[ARCH_INDEX(*sptr) - CHARSET_MIN]) {
            *dptr++ = *sptr++;
            if (--count <= 1) break;
      } else
            sptr++;
      *dptr = 0;
}

static void inc_new_count(unsigned int length, int count,
      char  *allchars,  char  *char1,  char2_table  char2,  chars_table
*chars)
{
      int pos, i, j;
      int size;

      log_event("- Expanding tables for length %d to character count
%d",
            length + 1, count + 1);

      size = count + 2;

      expand(char1, allchars, size);
      if (length)
            expand((*char2)[CHARSET_SIZE], allchars, size);
      for (pos = 0; pos <= (int)length - 2; pos++)
            expand((*chars[pos])[CHARSET_SIZE][CHARSET_SIZE],
                  allchars, size);

      for (i = 0; i < CHARSET_SIZE; i++) {
            if (length)
                  expand((*char2)[i], (*char2)[CHARSET_SIZE], size);
```

```
            for (j = 0; j < CHARSET_SIZE; j++)
            for (pos = 0; pos <= (int)length - 2; pos++) {
                  expand((*chars[pos])[i][j], (*chars[pos])
                        [CHARSET_SIZE][j], size);
                  expand((*chars[pos])[i][j], (*chars[pos])
                        [CHARSET_SIZE][CHARSET_SIZE], size);
            }
      }
}

static int inc_key_loop(int length, int fixed, int count,
      char *char1, char2_table char2, chars_table *chars
    , int inc_wu_size)
{
      char key_i[PLAINTEXT_BUFFER_SIZE];
      char key_e[PLAINTEXT_BUFFER_SIZE];
      char *key;
      char *chars_cache;
      int numbers_cache;
      int pos;

      key_i[length + 1] = 0;
      numbers[fixed] = count;

      chars_cache = NULL;

update_all:
      pos = 0;
update_ending:
      if (pos < 2) {
            if (pos == 0)
                  key_i[0] = char1[numbers[0]];
            if (length)
                  key_i[1]        =        (*char2)[key_i[0]        -
CHARSET_MIN][numbers[1]];
            pos = 2;
      }
      while (pos < length) {
            key_i[pos] = (*chars[pos - 2])
                  [ARCH_INDEX(key_i[pos - 2]) - CHARSET_MIN]
                  [ARCH_INDEX(key_i[pos - 1]) - CHARSET_MIN]
                  [numbers[pos]];
            pos++;
      }
      numbers_cache = numbers[length];
      if (pos == length) {
            chars_cache = (*chars[pos - 2])
                  [ARCH_INDEX(key_i[pos - 2]) - CHARSET_MIN]
                  [ARCH_INDEX(key_i[pos - 1]) - CHARSET_MIN];
update_last:
            key_i[length] = chars_cache[numbers_cache];
      }

      key = key_i;
```

```
      if (!ext_mode || !f_filter || ext_filter_body(key_i, key =
key_e))

      if (current_wu_size == inc_wu_size){
         fprintf(stderr,"Size finished = %d\n",current_wu_size);
         fprintf(stderr,"pos=%d\n",pos);
         fprintf(stderr,"key=%s\n",key);
         current_wu_size=0;
         return 1;
    }
//         fprintf(stderr,"key=%s, size = %d\n",key,current_wu_size);
    current_wu_size++;

      if (rec_compat) goto compat;

      pos = length;
      if (fixed < length) {
            if (++numbers_cache <= count) {
                  if (length >= 2) goto update_last;
                  numbers[length] = numbers_cache;
                  goto update_ending;
            }
            numbers[pos--] = 0;
            while (pos > fixed) {
                  if (++numbers[pos] <= count) goto update_ending;
                  numbers[pos--] = 0;
            }
      }
      while (pos-- > 0) {
            if (++numbers[pos] < count) goto update_ending;
            numbers[pos] = 0;
      }

      return 0;

compat:
      pos = 0;
      if (fixed) {
            if (++numbers[0] < count) goto update_all;
            if (!length && numbers[0] <= count) goto update_all;
            numbers[0] = 0;
            pos = 1;
            while (pos < fixed) {
                  if (++numbers[pos] < count) goto update_all;
                  numbers[pos++] = 0;
            }
      }
      while (++pos <= length) {
            if (++numbers[pos] <= count) goto update_all;
            numbers[pos] = 0;
      }

      return 0;
}
```

```
void do_incremental_crack(struct db_main *db, struct BATCH *b)
{
      char *charset;
      int min_length, max_length, max_count;
      char *extra;
      FILE *file;
      struct charset_header *header;
      char allchars[CHARSET_SIZE + 1];
      char char1[CHARSET_SIZE + 1];
      char2_table char2;
      chars_table chars[CHARSET_LENGTH - 2];
      unsigned char *ptr;
      unsigned int length, fixed, count;
      unsigned int real_count;
      unsigned int inc_wu_size;
    int last_length, last_count;
      int pos;
    char *mode;

//
strncpy(mode,batch_load_format(&b),sizeof(batch_load_format(&b)));
    if (!(strcmp(db->format->params.format_name,"NT LM DES")))
    {
        mode = "LanMan";
    }else
        mode ="All";


      if (!(charset = cfg_get_param(SECTION_INC, mode, "File"))) {
            fprintf(stderr,  "No  charset  defined  for  mode:  %s\n",
mode);
            error();
      }

      extra = cfg_get_param(SECTION_INC, mode, "Extra");

      if ((min_length = cfg_get_int(SECTION_INC, mode, "MinLen")) <
0)
            min_length = 0;
      if ((max_length = cfg_get_int(SECTION_INC, mode, "MaxLen")) <
0)
            max_length = CHARSET_LENGTH;

    max_count = cfg_get_int(SECTION_INC, mode, "CharCount");
    if ((inc_wu_size = (unsigned int)cfg_get_int(SECTION_BOINC, NULL
                    , "Inc_size")) < 0){
       fprintf(stderr,"Failed to get Inc_size\n");
       inc_wu_size=DEFAULT_INC_WU_SIZE;
    }
    fprintf(stderr,"INC_SIZE = %d\n",inc_wu_size);
      if (min_length > max_length) {
            fprintf(stderr, "MinLen = %d exceeds MaxLen = %d\n",
                  min_length, max_length);
```

```
            error();
      }

      if (max_length > CHARSET_LENGTH) {
            fprintf(stderr,"! MaxLen = %d exceeds the compile-time
limit of %d",
                  max_length, CHARSET_LENGTH);
            fprintf(stderr,
                  "\n"
                  "MaxLen = %d exceeds the compile-time limit of
%d\n\n"
                  "There're several good reasons why you probably
don't "
                  "need to raise it:\n"
                  "- many hash types don't support passwords "
                  "(or password halves) longer than\n"
                  "7 or 8 characters;\n"
                  "- you probably don't have sufficient statistical "
                  "information to generate a\n"
                  "charset file for lengths beyond 8;\n"
                  "-  the  limitation  applies  to  incremental  mode
only.\n",
                  max_length, CHARSET_LENGTH);
            error();
      }

      if (!(file = fopen(path_expand(charset), "rb")))
            pexit("fopen: %s", path_expand(charset));

      header = (struct charset_header *)mem_alloc(sizeof(*header));

      charset_read_header(file, header);
      if (ferror(file)) pexit("fread");

      if (feof(file) ||
          memcmp(header->version,   CHARSET_VERSION,   sizeof(header-
>version)) ||
          header->min  !=  CHARSET_MIN  ||  header->max  !=  CHARSET_MAX
||
          header->length != CHARSET_LENGTH ||
          header->count > CHARSET_SIZE || !header->count)
            inc_format_error(charset);

      fread(allchars, header->count, 1, file);
      if (ferror(file)) pexit("fread");
      if (feof(file)) inc_format_error(charset);

      allchars[header->count] = 0;
      if (extra)
            expand(allchars, extra, sizeof(allchars));
      real_count = strlen(allchars);

      if (max_count < 0) max_count = CHARSET_SIZE;
```

```
      if (min_length != max_length)
            fprintf(stderr,"- Lengths %d to %d, up to %d different
characters\n",
                  min_length, max_length, max_count);
      else
            fprintf(stderr,"-  Length  %d,  up  to  %d  different
characters\n",
                  min_length, max_count);

      if ((unsigned int)max_count > real_count) {
            fprintf(stderr,    "Warning:    only    %u    characters
available\n",
                  real_count);
      }

      if (header->length >= 2)
            char2 = (char2_table)mem_alloc(sizeof(*char2));
      else
            char2 = NULL;
      for (pos = 0; pos < (int)header->length - 2; pos++)
            chars[pos] = (chars_table)mem_alloc(sizeof(*chars[0]));

    b->s_loc=b->f_loc;
    b->s_length=b->f_length;
    b->s_fixed=b->f_fixed;
    b->s_count=b->f_count;
    b->s_num0=b->f_num0;
    b->s_num1=b->f_num1;
    b->s_num2=b->f_num2;
    b->s_num3=b->f_num3;
    b->s_num4=b->f_num4;
    b->s_num5=b->f_num5;
    b->s_num6=b->f_num6;
    b->s_num7=b->f_num7;
//            fprintf(stderr,"s_loc=%d,   s_length=%d,   s_fixed=%d,
s_count=%d\n"
//                  ,b->s_loc,b->s_length,b->s_fixed,b->s_count);

      rec_compat = 0;
      rec_entry = b->s_loc;

      memset(rec_numbers, 0, sizeof(rec_numbers));
    rec_numbers[0]=b->s_num0;
    rec_numbers[1]=b->s_num1;
    rec_numbers[2]=b->s_num2;
    rec_numbers[3]=b->s_num3;
    rec_numbers[4]=b->s_num4;
    rec_numbers[5]=b->s_num5;
    rec_numbers[6]=b->s_num6;
    rec_numbers[7]=b->s_num7;

      status_init(NULL, 0);
      entry=0;
/*
```

```
      rec_restore_mode(restore_state);
      rec_init(db, save_state);
*/
      ptr = header->order + (entry = rec_entry) * 3;
      memcpy(numbers, rec_numbers, sizeof(numbers));


//    crk_init(db, fix_state, NULL);

      last_count = last_length = -1;

      entry--;
      while (ptr < &header->order[sizeof(header->order) - 1])
     {
           entry++;
           length = *ptr++; fixed = *ptr++; count = *ptr++;

        if (entry == rec_entry)
        {
            if (length != b->s_length)
                fprintf(stderr,"Length does not match\n");
            if (fixed != b->s_fixed)
                fprintf(stderr,"Fixed does not match\n");
            if (count != b->s_count)
                fprintf(stderr,"Count does not match\n");
        }

           if (length >= CHARSET_LENGTH ||
                 fixed > length ||
                 count >= CHARSET_SIZE) inc_format_error(charset);

           if (entry != rec_entry)
                 memset(numbers, 0, sizeof(numbers));

        if (count >= real_count ||
                 (int)length  >=  db->format->params.plaintext_length
||
                 (fixed && !count)) continue;

           if ((int)length + 1 < min_length ||
                 (int)length >= max_length ||
                 (int)count >= max_count) continue;

           if ((int)length != last_length)
        {
                 inc_new_length(last_length = length,
                       header, file, charset, char1, char2, chars);
                 last_count = -1;
           }
           if ((int)count > last_count)
                 inc_new_count(length, last_count = count,
                       allchars, char1, char2, chars);

/*         if (!length && !min_length)
```

```
        {
                min_length = 1;
                if (crk_process_key("")) break;
           }
*/

//          log_event("- Trying length %d, fixed @%d, character count
%d",
//                length + 1, fixed + 1, count + 1);
           if  (inc_key_loop(length,  fixed,  count,  char1,  char2,
chars, inc_wu_size))
        {
//        fprintf(stderr,"Entry=%d,Length=%d, fixed=%d, count=%d\n"
//                ,entry,length,fixed,count);
            b->f_loc=entry;
            b->f_length=length;
            b->f_fixed=fixed;
            b->f_count=count;
            b->f_num0=numbers[0];
                 b->f_num1=numbers[1];
            b->f_num2=numbers[2];
            b->f_num3=numbers[3];
            b->f_num4=numbers[4];
            b->f_num5=numbers[5];
            b->f_num6=numbers[6];
            b->f_num7=numbers[7];
           if (gen_work(b))
            {
                 fprintf(stderr,"Failed   to   generate   work   for
INCREMENTAL MODE");
              b->mode_flag=BATCH_ERROR;
            }
            break;
        }
    }
//    error();
    if (ptr == &header->order[sizeof(header->order)-1])
          b->mode_flag=BATCH_COMPLETE;



//    crk_done();
//    rec_done(event_abort);

      for (pos = 0; pos < (int)header->length - 2; pos++)
           MEM_FREE(chars[pos]);
      MEM_FREE(char2);
      MEM_FREE(header);

      fclose(file);
}
```

H. JOHN_CONFIG.H

```
/*
 * This file is part of John the Ripper password cracker,
 * Copyright (c) 1996-2000 by Solar Designer
 */

/*
 * Configuration file loader.
 */

#ifndef _JOHN_CONFIG_H
#define _JOHN_CONFIG_H

/*
 * Parameter list entry.
 */
struct cfg_param {
      struct cfg_param *next;
      char *name, *value;
};

/*
 * Line list entry.
 */
struct cfg_line {
      struct cfg_line *next;
      char *data;
      int number;
};

/*
 * Main line list structure, head is used to start scanning the
list, while
 * tail is used to add new entries.
 */
struct cfg_list {
      struct cfg_line *head, *tail;
};

/*
 * Section list entry.
 */
struct cfg_section {
      struct cfg_section *next;
      char *name;
      struct cfg_param *params;
      struct cfg_list *list;
};

/*
 * Name of the currently loaded configuration file, or NULL for
none.
 */
```

```
extern char *cfg_name;

/*
 * Loads a configuration file, or does nothing if one is already
loaded.
 */
extern void cfg_init(char *name, int allow_missing);

/*
 * Searches for a section with the supplied name, and returns its
line list
 * structure, or NULL if the search fails.
 */
extern struct cfg_list *cfg_get_list(char *section, char
*subsection);

/*
 * Searches for a section with the supplied name and a parameter
within the
 * section, and returns the parameter's value, or NULL if not found.
 */
extern char *cfg_get_param(char *section, char *subsection, char
*param);

/*
 * Similar to the above, but does an atoi(). Returns -1 if not
found.
 */
extern int cfg_get_int(char *section, char *subsection, char
*param);

/*
 * Converts the value to boolean. Returns 0 (false) if not found.
 */
extern int cfg_get_bool(char *section, char *subsection, char
*param);

#endif
```

I. JOHN_CONFIG.C

```
/*
 * This file is part of John the Ripper password cracker,
 * Copyright (c) 1996-2002 by Solar Designer
 */

#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <ctype.h>
#include <errno.h>

#include "misc.h"
```

```
#include "params.h"
#include "path.h"
#include "memory.h"
#include "john_config.h"

char *cfg_name = NULL;
static struct cfg_section *cfg_database = NULL;

static char *trim(char *s)
{
	char *e;

	while (*s && (*s == ' ' || *s == '\t')) s++;
	if (!*s) return s;

	e = s + strlen(s) - 1;
	while (e >= s && (*e == ' ' || *e == '\t')) e--;
	*++e = 0;
	return s;
}

static void cfg_add_section(char *name)
{
	struct cfg_section *last;

	last = cfg_database;
	cfg_database = mem_alloc_tiny(
		sizeof(struct cfg_section), MEM_ALIGN_WORD);
	cfg_database->next = last;

	cfg_database->name = str_alloc_copy(name);
	cfg_database->params = NULL;

	if (!strncmp(name, "list.", 5)) {
		cfg_database->list = mem_alloc_tiny(
			sizeof(struct cfg_list), MEM_ALIGN_WORD);
		cfg_database->list->head = cfg_database->list->tail =
NULL;
	} else {
		cfg_database->list = NULL;
	}
}

static void cfg_add_line(char *line, int number)
{
	struct cfg_list *list;
	struct cfg_line *entry;

	entry = mem_alloc_tiny(sizeof(struct cfg_line),
MEM_ALIGN_WORD);
	entry->next = NULL;

	entry->data = str_alloc_copy(line);
	entry->number = number;
```

```
	list = cfg_database->list;
	if (list->tail)
		list->tail = list->tail->next = entry;
	else
		list->tail = list->head = entry;
}

static void cfg_add_param(char *name, char *value)
{
	struct cfg_param *current, *last;

	last = cfg_database->params;
	current = cfg_database->params = mem_alloc_tiny(
		sizeof(struct cfg_param), MEM_ALIGN_WORD);
	current->next = last;

	current->name = str_alloc_copy(name);
	current->value = str_alloc_copy(value);
}

static int cfg_process_line(char *line, int number)
{
	char *p;

	line = trim(line);
	if (!*line || *line == '#' || *line == ';') return 0;

	if (*line == '[') {
		if ((p = strchr(line, ']'))) *p = 0; else return 1;
		cfg_add_section(strlwr(trim(line + 1)));
	} else
	if (cfg_database && cfg_database->list) {
		cfg_add_line(line, number);
	} else
	if (cfg_database && (p = strchr(line, '='))) {
		*p++ = 0;
		cfg_add_param(strlwr(trim(line)), trim(p));
	} else {
		return 1;
	}

	return 0;
}

static void cfg_error(char *name, int number)
{
	fprintf(stderr, "Error in %s at line %d\n",
		path_expand(name), number);
	error();
}

void cfg_init(char *name, int allow_missing)
{
```

```
	FILE *file;
	char line[LINE_BUFFER_SIZE];
	int number;

	if (cfg_database) return;

	if (!(file = fopen(path_expand(name), "r"))) {
		if (allow_missing && errno == ENOENT) return;
		pexit("fopen: %s", path_expand(name));
	}

	number = 0;
	while (fgetl(line, sizeof(line), file))
	if (cfg_process_line(line, ++number)) cfg_error(name, number);

	if (ferror(file)) pexit("fgets");

	if (fclose(file)) pexit("fclose");

	cfg_name = str_alloc_copy(path_expand(name));
}

static struct cfg_section *cfg_get_section(char *section, char
*subsection)
{
	struct cfg_section *current;
	char *p1, *p2;

	if ((current = cfg_database))
	do {
		p1 = current->name; p2 = section;
		while (*p1 && *p1 == tolower(*p2)) {
			p1++; p2++;
		}
		if (*p2) continue;

		if ((p2 = subsection))
		while (*p1 && *p1 == tolower(*p2)) {
			p1++; p2++;
		}
		if (*p1) continue;
		if (p2) if (*p2) continue;

		return current;
	} while ((current = current->next));

	return NULL;
}

struct cfg_list *cfg_get_list(char *section, char *subsection)
{
	struct cfg_section *current;

	if ((current = cfg_get_section(section, subsection)))
```

```
            return current->list;

      return NULL;
}

char *cfg_get_param(char *section, char *subsection, char *param)
{
      struct cfg_section *current_section;
      struct cfg_param *current_param;
      char *p1, *p2;

      if ((current_section = cfg_get_section(section, subsection)))
      if ((current_param = current_section->params))
      do {
            p1 = current_param->name; p2 = param;
            while (*p1 && *p1 == tolower(*p2)) {
                  p1++; p2++;
            }
            if (*p1 || *p2) continue;

            return current_param->value;
      } while ((current_param = current_param->next));

      return NULL;
}

int cfg_get_int(char *section, char *subsection, char *param)
{
      char *s_value, *error;
      long l_value;

      if ((s_value = cfg_get_param(section, subsection, param))) {
            l_value = strtol(s_value, &error, 10);
            if (!*s_value || *error || (l_value & ~0x3FFFFFFFL))
                  return -1;
            return (int)l_value;
      }

      return -1;
}

int cfg_get_bool(char *section, char *subsection, char *param)
{
      char *value;

      if ((value = cfg_get_param(section, subsection, param)))
      switch (*value) {
      case 'y':
      case 'Y':
      case 't':
      case 'T':
      case '1':
            return 1;
      }
```

```
      return 0;
}
```

J. JOHN_DB.H

```
/*this file is the header file for boinc.c it contains
 *all structures, classes,  and functions dealing
 *with database interactions
 */
#ifndef _JOHN_DB_H
#define _JOHN_DB_H

//#include <cstdio>
//#include <vector>

#include <mysql.h>
#include <db_base.h>
#include <boinc_db.h>
#include "boinc.h"


extern DB_CONN john_db;


struct PASSWD {
    int id;
    char *login;
    char *passwd_hash;
    int UID;
    int GID;
    char *GECOS;
    char *home_dir;
    char *shell;
    char *password;
    char *format;
    int batch_id;

    void clear();
};

struct STATUS{
    int id;
    int pwid;
    int status_flag;
    int priority;
    unsigned long workunits_gen;
    unsigned long workunits_done;

    void clear();
};
```

```
struct PASSWD_ITEM {
      PASSWD pw;
      STATUS pw_stat;

      void clear();
      void parse(MYSQL_ROW&);
    PASSWD_ITEM& operator=(const PASSWD_ITEM &b);
};


void batch_equal(BATCH *t, BATCH &s);


class DB_PASSWD : public DB_BASE, public PASSWD{
public:
    DB_PASSWD(DB_CONN* p=0);
    int get_id();
    void db_print(char*);
    void db_parse(MYSQL_ROW &row);
    int check_dup(char*, char*,char*);
    void load_pw(char*,char*,char*,char*,char*,char*,char*,char*);

};

class DB_STATUS : public DB_BASE, public STATUS{
public:
    DB_STATUS(DB_CONN* p=0);
    int get_id();
    void db_print(char*);
    void db_parse(MYSQL_ROW &row);
    int check_flag(int flg);
    int update_stat_flag_to_working();
    void operator=(STATUS& r) {STATUS::operator=(r);}
};

class DB_BATCH : public DB_BASE, public BATCH{
public:
    DB_BATCH(DB_CONN* p=0);
    int get_id();
    void db_print(char*);
    void db_parse(MYSQL_ROW &row);
    void create_batch(int);
    int update_batch(BATCH &b);
    int load_format(BATCH &b, char **mode);
    int load_cipher(BATCH &b, char **cipher);
    int check_NOT_WORK(char*);
};
class DB_BATCH_SET:public DB_BASE_SPECIAL{
public:
    DB_BATCH_SET(DB_CONN* p=0);
    BATCH last_batch;
    int items_this_q;
```

```
    int load_set(std::vector<BATCH> &b);

};
class DB_PASSWD_ITEM_SET: public DB_BASE_SPECIAL{
public:
      DB_PASSWD_ITEM_SET(DB_CONN* p=0);
      PASSWD_ITEM last_item;
      int items_this_q;

      int  enumerate(std::vector<PASSWD_ITEM>&  items,  int  btch_id,
int stat_flg );
      int update_status(STATUS&);
      int update_passwd(PASSWD &pw);
      int get_passwd(int,char*);
      int update_pw(char*);
};



#endif
```

K. JOHN_DB.C

```
#include <cstdlib>
#include <cstring>
#include <ctime>
#include <unistd.h>
#include <stdio.h>
#include <sched_config.h>
#include <util.h>
#include <error_numbers.h>
#include "john_db.h"

extern "C" {
//#include "logger.h"
#include "memory.h"
//#include "misc.h"
#include "params.h"
}
using namespace std;

//SCHED_CONFIG config;
DB_CONN john_db;

//static struct PWLIST *passwd_list = NULL;

void PASSWD::clear() {memset(this, 0, sizeof(*this));}
void STATUS::clear() {memset(this, 0, sizeof(*this));}
void PASSWD_ITEM::clear() {
      memset(this, 0, sizeof(*this));
      pw.clear();
      pw_stat.clear();
}
```

```
DB_PASSWD::DB_PASSWD(DB_CONN* dc) :
      //DB_PASSWD("passwords", dc?dc:&john_db){}
      DB_BASE((const char*) "passwords", dc?dc:&john_db){}
DB_STATUS::DB_STATUS(DB_CONN* dc) :
      //DB_STATUS("status", dc?dc:&john_db){}
      DB_BASE("status", dc?dc:&john_db){}
DB_BATCH::DB_BATCH(DB_CONN* dc) :
      //DB_BATCH("batch", dc?dc:&john_db){}
      DB_BASE("batch", dc?dc:&john_db){}
DB_PASSWD_ITEM_SET::DB_PASSWD_ITEM_SET(DB_CONN* dc):
      //DB_PASSWD_ITEM_SET(dc?dc:&john_db){}
      DB_BASE_SPECIAL(dc?dc:&john_db){}
DB_BATCH_SET::DB_BATCH_SET(DB_CONN* dc):
    DB_BASE_SPECIAL(dc?dc:&john_db){}

int DB_PASSWD::get_id() {return id;}
int DB_STATUS::get_id() {return id;}
int DB_BATCH::get_id() {return id;}


void DB_PASSWD::db_print(char* buf){
   sprintf(buf,"id=%d, login='%s', passwd_hash='%s',"
      "userid='%d',      groupid='%d',      GECOS='%s',home_dir='%s',
shell='%s',"
      "password='%s',format='%s',batch_id='%d'",      id,      login,
passwd_hash, UID, GID,
      GECOS, home_dir, shell, password, format,batch_id);
}



int DB_PASSWD::check_dup(char *l, char *pw_hash,char *f){
      char query[MAX_QUERY_LEN];
      int retval;
      MYSQL_RES *res;
      MYSQL_ROW row;

      sprintf(query,"Select * from passwords where login='%s'"
                  "   and   passwd_hash='%s'   and   format='%s'",   l,
pw_hash,f);
      retval=db->do_query(query);
      if (retval) return mysql_errno(db->mysql);
      res = mysql_store_result(db->mysql);
      if (!res) return mysql_errno(db->mysql);
      row = mysql_fetch_row(res);
      if (row)
            return 0;
      return 1;
}
int DB_BATCH::check_NOT_WORK(char *form){
    char query[MAX_QUERY_LEN];
    int retval;
    MYSQL_RES *res;
```

```
    MYSQL_ROW row;
    PASSWD_ITEM pi;
    pi.clear();
//    fprintf(stderr,"IN Check_NOT_WORKED\n");
    sprintf(query, "Select * from passwords AS pw"
                   " LEFT JOIN status AS stat ON pw.id = stat.pwid"
                   "     WHERE     pw.batch_id     !=     '%d'     and
stat.status_flag='%d'"
                  " and pw.format='%s' LIMIT 1 "
                  ,BATCH_NOT_SET,NOT_WORKED,form);
    retval=db->do_query(query);
//    fprintf(stderr,"retval=%d\n",retval);
    if (retval) return mysql_errno(db->mysql);
    res= mysql_store_result(db->mysql);
    if (!res) return mysql_errno(db->mysql);
    row = mysql_fetch_row(res);
    if (row){
        pi.parse(row);
//                                    fprintf(stderr,"b_ID    %d:form
%s\n",pi.pw.batch_id,pi.pw.format);
        return pi.pw.batch_id;
    }
    return -1;
}
void DB_PASSWD::db_parse(MYSQL_ROW &r){
      int i=0;
      clear();
      id=atol(r[i++]);
      login=r[i++];
      passwd_hash=r[i++];
      UID=atol(r[i++]);
      GID=atol(r[i++]);
      GECOS=r[i++];
      home_dir=r[i++];
      shell=r[i++];
      password=r[i++];
      format=r[i++];
    batch_id=atoi(r[i++]);
}

void DB_STATUS::db_print(char* buf){
   sprintf(buf,
        "id='%d', pwid='%d', status_flag='%d', priority='%d',"
        "workunits_gen='%d', workunits_done='%d'", id, pwid,
      status_flag, priority, workunits_gen, workunits_done);
}

void DB_STATUS::db_parse(MYSQL_ROW &r){
      int i=0;
      clear();
      id=atol(r[i++]);
      pwid=atol(r[i++]);
      status_flag=atol(r[i++]);
      priority=atol(r[i++]);
```

```
      workunits_gen=atol(r[i++]);
      workunits_done=atol(r[i++]);
}
/*this function returns true if an object
in the status table has a flag=flg*/
int DB_STATUS::check_flag(int flg){
      char query[MAX_QUERY_LEN];
      int retval;
      MYSQL_ROW row;
      MYSQL_RES *res;
      sprintf(query,"Select * from passwords as pw LEFT JOIN status
as stat ON "
                  "pw.id = stat.pwid where stat.status_flag = '%d'
and "
                  "pw.batch_id !='-1' limit 1", flg );
      retval = db->do_query(query);
      if (retval) return mysql_errno(db->mysql);

      res = mysql_store_result(db->mysql);
      if (!res) return mysql_errno(db->mysql);

      row = mysql_fetch_row(res);
      if (!row){
            mysql_free_result(res);
            return 0;
      }
      return 1;
}
int DB_STATUS::update_stat_flag_to_working(){
    char query[MAX_QUERY_LEN];
    sprintf(query,"update status set status_flag ='%d'"
            " where status_flag ='%d'",WORKING,NOT_WORKED);
    return db->do_query(query);
}
void DB_BATCH::db_print(char *buf){
      sprintf(buf," id='%d', mode_flag='%d', batch_priority='%d',"
        "s_loc='%d', s_length='%d', s_fixed='%d', s_count='%d',"
        "s_num0='%d', s_num1='%d', s_num2='%d', s_num3='%d',"
        "s_num4='%d', s_num5='%d', s_num6='%d', s_num7='%d',"
        "f_loc='%d', f_length='%d', f_fixed='%d', f_count='%d',"
        "f_num0='%d', f_num1='%d', f_num2='%d', f_num3='%d',"
        "f_num4='%d', f_num5='%d', f_num6='%d', f_num7='%d'"
        ,id,mode_flag,batch_priority,s_loc,s_length,s_fixed,s_count
        ,s_num0,s_num1,s_num2,s_num3,s_num4,s_num5,s_num6,s_num7

,f_loc,f_length,f_fixed,f_count,f_num0,f_num1,f_num2,f_num3,f_num4
        ,f_num5,f_num6,f_num7);

}

void DB_BATCH::db_parse(MYSQL_ROW &r){
    int i=0;
    id=atoi(r[i++]);
    mode_flag=atoi(r[i++]);
```

```
    batch_priority =atoi(r[i++]);
    s_loc=atoi(r[i++]);
    s_length=atoi(r[i++]);
    s_fixed=atoi(r[i++]);
    s_count=atoi(r[i++]);
    s_num0=atoi(r[i++]);
    s_num1=atoi(r[i++]);
    s_num2=atoi(r[i++]);
    s_num3=atoi(r[i++]);
    s_num4=atoi(r[i++]);
    s_num5=atoi(r[i++]);
    s_num6=atoi(r[i++]);
    s_num7=atoi(r[i++]);
    f_loc=atoi(r[i++]);
    f_length=atoi(r[i++]);
    f_fixed=atoi(r[i++]);
    f_count=atoi(r[i++]);
    f_num0=atoi(r[i++]);
    f_num1=atoi(r[i++]);
    f_num2=atoi(r[i++]);
    f_num3=atoi(r[i++]);
    f_num4=atoi(r[i++]);
    f_num5=atoi(r[i++]);
    f_num6=atoi(r[i++]);
    f_num7=atoi(r[i++]);
}

void DB_BATCH::create_batch(int p){
//    fprintf(stderr,"New batch created\n");
    id=0;
    mode_flag=SINGLE_MODE;
    batch_priority=p;
    s_loc=0;
    s_length=3;
    s_fixed=0;
    s_count=0;
    s_num0=0;
    s_num1=0;
    s_num2=0;
    s_num3=0;
    s_num4=0;
    s_num5=0;
    s_num6=0;
    s_num7=0;
    f_loc=0;
    f_length=0;
    f_fixed=0;
    f_count=0;
    f_num0=0;
    f_num1=0;
    f_num2=0;
    f_num3=0;
    f_num4=0;
    f_num5=0;
```

```
    f_num6=0;
    f_num7=0;
    if(insert()){
           fprintf(stderr,"failed to insert Batch\n");
    }
    id=db->insert_id();
}

void batch_parse(BATCH &b,MYSQL_ROW &r){
    int i=0;
    b.id=atoi(r[i++]);
    b.mode_flag=atoi(r[i++]);
    b.batch_priority =atoi(r[i++]);
    b.s_loc=atoi(r[i++]);
    b.s_length=atoi(r[i++]);
    b.s_fixed=atoi(r[i++]);
    b.s_count=atoi(r[i++]);
    b.s_num0=atoi(r[i++]);
    b.s_num1=atoi(r[i++]);
    b.s_num2=atoi(r[i++]);
    b.s_num3=atoi(r[i++]);
    b.s_num4=atoi(r[i++]);
    b.s_num5=atoi(r[i++]);
    b.s_num6=atoi(r[i++]);
    b.s_num7=atoi(r[i++]);
    b.f_loc=atoi(r[i++]);
    b.f_length=atoi(r[i++]);
    b.f_fixed=atoi(r[i++]);
    b.f_count=atoi(r[i++]);
    b.f_num0=atoi(r[i++]);
    b.f_num1=atoi(r[i++]);
    b.f_num2=atoi(r[i++]);
    b.f_num3=atoi(r[i++]);
    b.f_num4=atoi(r[i++]);
    b.f_num5=atoi(r[i++]);
    b.f_num6=atoi(r[i++]);
    b.f_num7=atoi(r[i++]);

}
int DB_BATCH_SET::load_set(std::vector<BATCH> &b){
      int retval;
      char query[MAX_QUERY_LEN];
      MYSQL_ROW row;
//    fprintf(stderr,"IN load_set\n");
      if (!cursor.active){
      sprintf(query, "Select * from batch WHERE mode_flag != '%d' "
                ,BATCH_COMPLETE);

//          fprintf(stderr,"%s\n",query);
        retval = db->do_query(query);

            if (retval) return mysql_errno(db->mysql);

            cursor.rp = mysql_store_result(db->mysql);
```

```
            if (!cursor.rp) return mysql_errno(db->mysql);
            cursor.active = true;
            if(cursor.active)
                  fprintf(stderr,"cursor active\n");

            row = mysql_fetch_row(cursor.rp);
            if (!row){
                  mysql_free_result(cursor.rp);
                  cursor.active = false;
                  return 1;
            }
//         last_item.clear();
//          last_item.parse(row);
            items_this_q = 0;
      }
    b.clear();
      while (true) {
        BATCH new_b;
        batch_parse(new_b,row);
        b.push_back(new_b);
        items_this_q++;
            row = mysql_fetch_row(cursor.rp);
            if (!row) {
                  mysql_free_result(cursor.rp);
                  cursor.active = false;
                  return 0;
            }
    }

      return 0;
}



void PASSWD_ITEM::parse(MYSQL_ROW &r){
/*fill in the fields I want to parse from
the rows returned in the enumerate function*/
    int i=0;
    clear();
    pw.id=atoi(r[i++]);
    pw.login=r[i++];
    pw.passwd_hash=r[i++];
    pw.UID=atoi(r[i++]);
    pw.GID=atoi(r[i++]);
    pw.GECOS=r[i++];
    pw.home_dir=r[i++];
    pw.shell=r[i++];
    pw.password=r[i++];
    pw.format=r[i++];
    pw.batch_id=atoi(r[i++]);
    pw_stat.id=atoi(r[i++]);
    pw_stat.pwid=atoi(r[i++]);
    pw_stat.status_flag=atoi(r[i++]);
    pw_stat.priority=atoi(r[i++]);
```

```
    pw_stat.workunits_gen=atoi(r[i++]);
    pw_stat.workunits_done=atoi(r[i++]);
}
PASSWD_ITEM&  PASSWD_ITEM::operator=(const PASSWD_ITEM &b){
      pw.id=b.pw.id;
      pw.login = new char[254];
      strcpy(pw.login,b.pw.login);
      pw.passwd_hash = new char[254];
      strcpy(pw.passwd_hash,b.pw.passwd_hash);
      pw.UID=b.pw.UID;
      pw.GID=b.pw.GID;
      pw.GECOS = new char[254];
      strcpy(pw.GECOS,b.pw.GECOS);
      pw.home_dir = new char[254];
      strcpy(pw.home_dir, b.pw.home_dir);
      pw.shell = new char[254];
      strcpy(pw.shell,b.pw.shell);
      pw.password = new char[254];
      strcpy(pw.password,b.pw.password);
      pw.format = new char[254];
      strcpy(pw.format,b.pw.format);
    pw.batch_id=b.pw.batch_id;
      pw_stat.id=b.pw_stat.id;
      pw_stat.pwid=b.pw_stat.pwid;
      pw_stat.status_flag=b.pw_stat.status_flag;
      pw_stat.priority=b.pw_stat.priority;
      pw_stat.workunits_gen=b.pw_stat.workunits_gen;
      pw_stat.workunits_done=b.pw_stat.workunits_done;
      return *this;

}
void batch_equal(BATCH *t, BATCH &s){
   t->id=s.id;
   t->mode_flag=s.mode_flag;
   t->batch_priority=s.batch_priority;
   t->s_loc=s.s_loc;//s=start ... Entry or Rule #
   t->s_length=s.s_length;
   t->s_fixed=s.s_fixed;
   t->s_count=s.s_count;
   t->s_num0=s.s_num0;
   t->s_num1=s.s_num1;
   t->s_num2=s.s_num2;
   t->s_num3=s.s_num3;
   t->s_num4=s.s_num4;
   t->s_num5=s.s_num5;
   t->s_num6=s.s_num6;
   t->s_num7=s.s_num7;
   t->f_loc=s.f_loc;
   t->f_length=s.f_length;
   t->f_fixed=s.f_fixed;
   t->f_count=s.f_count;
   t->f_num0=s.f_num0;
   t->f_num1=s.f_num1;
   t->f_num2=s.f_num2;
```

```
   t->f_num3=s.f_num3;
   t->f_num4=s.f_num4;
   t->f_num5=s.f_num5;
   t->f_num6=s.f_num6;
   t->f_num7=s.f_num7;

}
void  DB_PASSWD::load_pw(char*  l,  char  *pw_hash,  char  *gecos,char
*home,
                 char *uid, char *gid, char *SHELL, char *form){
      int retval;
      DB_STATUS stat;

      id=0;
//        fprintf(stderr,"id:%d\n",id);
//        fprintf(stderr,"passed in var:%s\n",l);
//      strcpy2(login,login);
        login=l;
//        fprintf(stderr,"login:%s\n",login);
        passwd_hash=pw_hash;
//        fprintf(stderr,"password_hash:%s\n",passwd_hash);
        UID=atoi(uid);
        GID=atoi(gid);
        GECOS=gecos;
        home_dir=home;
        shell=SHELL;
          password="NO_PASSWORD";
          format=form;
        batch_id=BATCH_NOT_SET;
        retval=insert();
/*        fprintf(stderr,"%d:%s:%s:%d:%d:%s:%s:%s:%s\n", id,login,
                       passwd_hash,UID,GID,GECOS,
                       home_dir,shell,format);
*/
        if(retval){
                //log_event("Failed     to     insert     pw     into
%s",table_name);
                fprintf(stderr,"Failed    to    insert    pw    into
%s",table_name);
                //error();
        }
        //this retrieves the last thing inserted to the DB
        id=db->insert_id();
        stat.clear();
        stat.id=0;
        stat.pwid=id;
        stat.status_flag=NOT_WORKED;
        stat.priority=DEFAULT_PRIORITY;
        stat.workunits_gen=0;
        stat.workunits_done=0;
        retval=stat.insert();
        if (retval){
              //    log_event("Failed    to    insert    stat    into
%s",stat.table_name);
```

```
                fprintf(stderr,"Failed    to    insert    stat    into
%s",stat.table_name);
                //error();
      }


}
// useful, but didn't use
int DB_PASSWD_ITEM_SET::get_passwd(int pw_id,char *pw_hash){
      char query[MAX_QUERY_LEN];
      int retval;
      MYSQL_RES *res;
      MYSQL_ROW row;
      sprintf(query,"Select * from passwords as pw "
                "left join status as st on pw.id=st.pwid where"
                "    pw.id    =    '%d'    and    pw.passwd_hash    =
'%s'",pw_id,pw_hash);
      retval = db->do_query(query);
      if (retval) return mysql_errno(db->mysql);
      res = mysql_store_result(db->mysql);
      if (!res) return mysql_errno(db->mysql);
      row = mysql_fetch_row(res);
      last_item.parse(row);
      return 0;

}
// didn't use ... look at more before delete
int DB_PASSWD_ITEM_SET::update_pw(char *plain){
      char *pass;
      pass = str_alloc_copy(plain);
      if (!(strncmp(pass,"NO_PASSWORD",11))){
            return 0;
      }else{
            last_item.pw.password = pass;
            last_item.pw_stat.status_flag = CRACKED;
        last_item.pw_stat.workunits_done++;
            if (update_passwd(last_item.pw)) return 1;
            if (update_status(last_item.pw_stat)) return 1;
      }
      return 0;
}
int DB_BATCH::load_format(BATCH &b, char **mode){
    int retval;
    char query[MAX_QUERY_LEN];
    MYSQL_ROW row;
    MYSQL_RES *res;

    sprintf(query,"Select format from passwords where batch_id='%d'
limit 1"
             ,b.id);
    retval = db->do_query(query);

    if (retval) return mysql_errno(db->mysql);
    res=mysql_store_result(db->mysql);
```

```
    if (!res) return mysql_errno(db->mysql);
    row=mysql_fetch_row(res);
    if (!row){
        mysql_free_result(res);
        return 1;
    }
    *mode= new char[128];
    strcpy(*mode, row[0]);
    mysql_free_result(res);
    return 0;
}
int DB_BATCH::load_cipher(BATCH &b, char **cipher){
    int retval;
    char query[MAX_QUERY_LEN];
    MYSQL_ROW row;
    MYSQL_RES *res;
 //   fprintf(stderr,"Top load_cipher\n");
    sprintf(query,"Select   passwd_hash   from   passwords   where
batch_id='%d' "
            "limit 1",b.id);

    retval = db->do_query(query);
    if (retval) return mysql_errno(db->mysql);
    res=mysql_store_result(db->mysql);
    if (!res) return mysql_errno(db->mysql);
    row=mysql_fetch_row(res);

    if (!row){
        mysql_free_result(res);
        return 1;
    }
    *cipher=new char[128];
    strcpy(*cipher, row[0]);
    mysql_free_result(res);
    return 0;

}
/*This function Queries the John DB for all passwords and there
status by using a Left outer Join on the tables passwords and
status.
Then the attributes asked for are stored into a VECTOR of
PASSWD_ITEMs
This vector will be used by the work generator to create workunits
for
a particular password hash*/
int          DB_PASSWD_ITEM_SET::enumerate(std::vector<PASSWD_ITEM>&
items,int btch_id, int stat_flg){
      int retval;
      char query[MAX_QUERY_LEN];
      MYSQL_ROW row;

      fprintf(stderr,"In enum\n");
      if (!cursor.active){
      sprintf(query, "Select * from passwords AS pw "
```

```
                "  LEFT JOIN status AS stat ON pw.id = stat.pwid "
                "         WHERE    pw.batch_id    =    '%d'    and
stat.status_flag='%d' "
               ,btch_id,stat_flg);

//         fprintf(stderr,"%s\n",query);
        retval = db->do_query(query);

           if (retval) return mysql_errno(db->mysql);

           cursor.rp = mysql_store_result(db->mysql);
           if (!cursor.rp) return mysql_errno(db->mysql);
           cursor.active = true;
           if(cursor.active)
                 fprintf(stderr,"cursor active\n");

           row = mysql_fetch_row(cursor.rp);
           if (!row){
                 mysql_free_result(cursor.rp);
                 cursor.active = false;
                 return 1;
           }
//        last_item.clear();
//         last_item.parse(row);
           items_this_q = 0;
      }
      items.clear();
      while (true) {
        last_item.clear();
        last_item.parse(row);
        items.push_back(last_item);
        items_this_q++;
           fprintf(stderr,"%s:%s:%s:%d:%d:%d\n",last_item.pw.login,
                       last_item.pw.passwd_hash,
                last_item.pw.password,
                       last_item.pw_stat.status_flag,
                       last_item.pw_stat.priority,
                       last_item.pw.batch_id);
           row = mysql_fetch_row(cursor.rp);
           if (!row) {
                 mysql_free_result(cursor.rp);
                 cursor.active = false;
                 return 0;
           }
    }

      return 0;
}

//for updating a STATUS object
//used by workgenerator
int DB_BATCH::update_batch(BATCH &b){
    char query[MAX_QUERY_LEN];
    sprintf(query,"update batch set"
```

```
        " mode_flag='%d', batch_priority='%d',"
        "s_loc='%d', s_length='%d', s_fixed='%d', s_count='%d',"
        "s_num0='%d', s_num1='%d', s_num2='%d', s_num3='%d',"
        "s_num4='%d', s_num5='%d', s_num6='%d', s_num7='%d',"
        "f_loc='%d', f_length='%d', f_fixed='%d', f_count='%d',"
        "f_num0='%d', f_num1='%d', f_num2='%d', f_num3='%d',"
        "f_num4='%d', f_num5='%d', f_num6='%d', f_num7='%d'"
        "where
id='%d'",b.mode_flag,b.batch_priority,b.s_loc,b.s_length

,b.s_fixed,b.s_count,b.s_num0,b.s_num1,b.s_num2,b.s_num3,b.s_num4

,b.s_num5,b.s_num6,b.s_num7,b.f_loc,b.f_length,b.f_fixed,b.f_count

,b.f_num0,b.f_num1,b.f_num2,b.f_num3,b.f_num4,b.f_num5,b.f_num6
        ,b.f_num7,b.id);
 //   fprintf(stderr,"%s\n",query);
    return db->do_query(query);

}
int DB_PASSWD_ITEM_SET::update_status(STATUS &stat){
      char query[MAX_QUERY_LEN];
      sprintf(query,"update     status     set     status_flag='%d',
priority='%d',"
           "workunits_gen='%d', workunits_done='%d' where id='%d'"
        , stat.status_flag, stat.priority, stat.workunits_gen
        , stat.workunits_done,stat.id);
      return db->do_query(query);
}

//for updating passwords when cracked
//used by validator...
int DB_PASSWD_ITEM_SET::update_passwd(PASSWD &pw){
        char query[MAX_QUERY_LEN];
    fprintf(stderr,"password=%s, batch_id=%d, pwid=%d\n",pw.password
           , pw.batch_id, pw.id);
      sprintf(query,"update     passwords     set     password='%s',
batch_id='%d' where "
           "id='%d'", pw.password, pw.batch_id, pw.id);
      fprintf(stderr,"%s\n",query);
      return db->do_query(query);
}
```

L. OPTIONS.H

```
/*
 * This file is part of John the Ripper password cracker,
 * Copyright (c) 1996-98,2003 by Solar Designer
 */

/*
 * John's command line options definition.
 */
```

```
#ifndef _JOHN_OPTIONS_H
#define _JOHN_OPTIONS_H

#include "list.h"
#include "loader.h"
#include "getopt.h"

/*
 * Option flags bitmasks.
 */
/* An action requested */
#define FLG_ACTION                0x00000001
/* Password files specified */
#define FLG_PASSWD                0x00000002
/* An option supports password files */
#define FLG_PWD_SUP               0x00000004
/* An option requires password files */
#define FLG_PWD_REQ               (0x00000008 | FLG_PWD_SUP)
/* Some option that doesn't have its own flag is specified */
#define FLG_NONE            0x00000010
/* A cracking mode enabled */
#define FLG_CRACKING_CHK          0x00000020
#define FLG_CRACKING_SUP          0x00000040
#define FLG_CRACKING_SET \
      (FLG_CRACKING_CHK    |    FLG_CRACKING_SUP    |    FLG_ACTION    |
FLG_PWD_REQ)
/* Wordlist mode enabled, options.wordlist is set to the file name
or NULL
 * if reading from stdin. */
#define FLG_WORDLIST_CHK          0x00000080
#define FLG_WORDLIST_SET          (FLG_WORDLIST_CHK                |
FLG_CRACKING_SET)
/* Wordlist mode enabled, reading from stdin */
#define FLG_STDIN_CHK             0x00000100
#define FLG_STDIN_SET             (FLG_STDIN_CHK | FLG_WORDLIST_SET)
/* Wordlist rules enabled */
#define FLG_RULES           0x00000200
/* "Single crack" mode enabled */
#define FLG_SINGLE_CHK            0x00000400
#define FLG_SINGLE_SET            (FLG_SINGLE_CHK                  |
FLG_CRACKING_SET)
/* Incremental mode enabled */
#define FLG_INC_CHK               0x00000800
#define FLG_INC_SET               (FLG_INC_CHK | FLG_CRACKING_SET)
/* External mode or word filter enabled */
#define FLG_EXTERNAL_CHK          0x00001000
#define FLG_EXTERNAL_SET \
      (FLG_EXTERNAL_CHK    |    FLG_ACTION    |    FLG_CRACKING_SUP    |
FLG_PWD_SUP)
/* Batch cracker */
#define FLG_BATCH_CHK             0x00004000
#define FLG_BATCH_SET             (FLG_BATCH_CHK | FLG_CRACKING_SET)
/* Stdout mode */
```

```
#define FLG_STDOUT                0x00008000
/* Restoring an interrupted session */
#define FLG_RESTORE_CHK                 0x00010000
#define FLG_RESTORE_SET                 (FLG_RESTORE_CHK           |
FLG_ACTION)
/* A session name is set */
#define FLG_SESSION               0x00020000
/* Print status of a session */
#define FLG_STATUS_CHK            0x00040000
#define FLG_STATUS_SET            (FLG_STATUS_CHK | FLG_ACTION)
/* Make a charset */
#define FLG_MAKECHR_CHK                 0x00100000
#define FLG_MAKECHR_SET \
      (FLG_MAKECHR_CHK | FLG_ACTION | FLG_PWD_SUP)
/* Show cracked passwords */
#define FLG_SHOW_CHK              0x00200000
#define FLG_SHOW_SET \
      (FLG_SHOW_CHK | FLG_ACTION | FLG_PWD_REQ)
/* Perform a benchmark */
#define FLG_TEST_CHK              0x00400000
#define FLG_TEST_SET \
      (FLG_TEST_CHK | FLG_CRACKING_SUP | FLG_ACTION)
/* Passwords per salt requested */
#define FLG_SALTS           0x01000000
/* Ciphertext format forced */
#define FLG_FORMAT                0x02000000
/* Memory saving enabled */
#define FLG_SAVEMEM               0x04000000
/* Application Name Set*/
#define FLG_APP                   0x10000000

/*
 * Structure with option flags and all the parameters.
 */
struct options_main {
/* Option flags */
      opt_flags flags;

/* Password files */
      struct list_main *passwd;

/* Password file loader options */
      struct db_options loader;

/* Session name */
      char *session;

/* Ciphertext format name */
      char *format;

/* Wordlist file name */
      char *wordlist;

/* Charset file name */
```

```
      char *charset;

/* External mode or word filter name */
      char *external;
/* Application Name */
      char *app_name;

/* Maximum plaintext length for stdout mode */
      int length;
};

extern struct options_main options;

/*
 * Initializes the options structure.
 */
extern void opt_init(int argc, char **argv);

#endif
```

M. OPTIONS.C

```
/*
 * This file is part of John the Ripper password cracker,
 * Copyright (c) 1996-2005 by Solar Designer
 */

#include <stdio.h>
#include <stdlib.h>
#include <string.h>

#include "arch.h"
#include "misc.h"
#include "params.h"
#include "memory.h"
#include "list.h"
#include "loader.h"
#include "logger.h"
#include "status.h"
#include "recovery.h"
#include "options.h"


struct options_main options;

static struct opt_entry opt_list[] = {
      {"", FLG_PASSWD, 0, 0, 0, OPT_FMT_ADD_LIST, &options.passwd},
      {"app",FLG_APP,0,0,0,OPT_FMT_STR_ALLOC, &options.app_name},
      {"single", FLG_SINGLE_SET, FLG_CRACKING_CHK},
      {"wordlist", FLG_WORDLIST_SET, FLG_CRACKING_CHK,
            0, OPT_REQ_PARAM, OPT_FMT_STR_ALLOC, &options.wordlist},
      {"rules",      FLG_RULES,      FLG_RULES,      FLG_WORDLIST_CHK,
FLG_STDIN_CHK},
```

```
      {"incremental", FLG_INC_SET, FLG_CRACKING_CHK,
            0, 0, OPT_FMT_STR_ALLOC, &options.charset},
/*    {"make-charset", FLG_MAKECHR_SET, FLG_MAKECHR_CHK,
            0, FLG_CRACKING_CHK | FLG_SESSION | OPT_REQ_PARAM,
            OPT_FMT_STR_ALLOC, &options.charset},
      {"show", FLG_SHOW_SET, FLG_SHOW_CHK,
            0, FLG_CRACKING_SUP | FLG_MAKECHR_CHK},
      {"test", FLG_TEST_SET, FLG_TEST_CHK,
            0, ~FLG_TEST_SET & ~FLG_FORMAT & ~FLG_SAVEMEM &
            ~OPT_REQ_PARAM},
      {"users", FLG_NONE, 0, FLG_PASSWD, OPT_REQ_PARAM,
            OPT_FMT_ADD_LIST_MULTI, &options.loader.users},
      {"groups", FLG_NONE, 0, FLG_PASSWD, OPT_REQ_PARAM,
            OPT_FMT_ADD_LIST_MULTI, &options.loader.groups},
      {"shells", FLG_NONE, 0, FLG_PASSWD, OPT_REQ_PARAM,
            OPT_FMT_ADD_LIST_MULTI, &options.loader.shells},
      {"salts", FLG_SALTS, FLG_SALTS, FLG_PASSWD, OPT_REQ_PARAM,
            "%d", &options.loader.min_pps},
      {"format", FLG_FORMAT, FLG_FORMAT,
            FLG_CRACKING_SUP,
            FLG_MAKECHR_CHK | FLG_STDOUT | OPT_REQ_PARAM,
            OPT_FMT_STR_ALLOC, &options.format},
      {"save-memory", FLG_SAVEMEM, FLG_SAVEMEM, 0, OPT_REQ_PARAM,
            "%u", &mem_saving_level},*/
      {NULL}
};

#if DES_BS
/* nonstd.c and parts of x86-mmx.S aren't mine */
#define JOHN_COPYRIGHT \
      "Solar Designer and others"
#else
#define JOHN_COPYRIGHT \
      "Solar Designer"
#endif
//changed
#define JOHN_BOINC_USAGE \
"Distributed John the Ripper password cracker - Work Generator ,
version \
 " JOHN_BOINC_VERSION "\n" \
"Copyright (c) 1996-2005 by " JOHN_COPYRIGHT "\n" \
"Homepage: \n" \
"\n" \
"Usage: %s [OPTIONS] [PASSWORD-FILES]\n" \
"--single                    \"single crack\" mode\n" \
"--app=NAME          Application Name\n" \
"--wordlist=FILE --stdin     wordlist mode, read words from FILE or
stdin\n" \
"--rules                     enable word mangling rules for wordlist
mode\n" \
"--incremental[=MODE]          \"incremental\" mode [using section
MODE]\n" /*\
"--make-charset=FILE               make a charset, FILE will be
overwritten\n" \
```

```
"--show                     show cracked passwords\n" \
"--test                     perform a benchmark\n" \
"--users=[-]LOGIN|UID[,..]    [do  not]  load  this  (these)  user(s)
only\n" \
"--groups=[-]GID[,..]            load  users  [not]  of  this  (these)
group(s) only\n" \
"--shells=[-]SHELL[,..]         load  users  with[out]  this  (these)
shell(s) only\n" \
"--salts=[-]COUNT              load salts with[out] at least COUNT
passwords " \
      "only\n" \
"--format=NAME              force ciphertext format NAME: " \
      "DES/BSDI/MD5/BF/AFS/LM\n" \
"--save-memory=LEVEL        enable memory saving, at LEVEL 1..3\n"
*/
void opt_init(int argc, char **argv)
{
      if (!argv[0]) error();

      if (!argv[1]) {
            printf(JOHN_BOINC_USAGE, argv[0]);
            exit(0);
      }

      memset(&options, 0, sizeof(options));

      list_init(&options.passwd);

      options.loader.flags = DB_LOGIN;
      list_init(&options.loader.users);
      list_init(&options.loader.groups);
      list_init(&options.loader.shells);

      options.length = -1;

      opt_process(opt_list, &options.flags, argv);

      if ((options.flags &
          (FLG_EXTERNAL_CHK  |  FLG_CRACKING_CHK  |  FLG_MAKECHR_CHK))
==
          FLG_EXTERNAL_CHK)
            options.flags |= FLG_CRACKING_SET;

      if (!(options.flags & FLG_ACTION))
            options.flags |= FLG_BATCH_SET;

      opt_check(opt_list, options.flags, argv);

      if (options.session)
            rec_name = options.session;

      if (options.flags & FLG_RESTORE_CHK) {
            rec_restore_args(1);
            return;
```

```
	}

	if (options.flags & FLG_STATUS_CHK) {
		rec_restore_args(0);
		options.flags |= FLG_STATUS_SET;
		status_init(NULL, 1);
		status_print();
		exit(0);
	}

	if (options.flags & FLG_SALTS)
	if (options.loader.min_pps < 0) {
		options.loader.max_pps = -1 - options.loader.min_pps;
		options.loader.min_pps = 0;
	}

	if (options.length < 0)
		options.length = PLAINTEXT_BUFFER_SIZE - 3;
	else
	if (options.length < 1 || options.length >
PLAINTEXT_BUFFER_SIZE - 3) {
		fprintf(stderr, "Invalid plaintext length requested\n");
		error();
	}

	if (options.flags & FLG_STDOUT) options.flags &= ~FLG_PWD_REQ;

	if ((options.flags & (FLG_PASSWD | FLG_PWD_REQ)) ==
FLG_PWD_REQ) {
		fprintf(stderr, "Password files required, "
			"but none specified\n");
		error();
	}

	if ((options.flags & (FLG_PASSWD | FLG_PWD_SUP)) ==
FLG_PASSWD) {
		fprintf(stderr, "Password files specified, "
			"but no option would use them\n");
		error();
	}

	rec_argc = argc; rec_argv = argv;
}
```

N. PARAMS.H

```
/*
 * This file is part of John the Ripper password cracker,
 * Copyright (c) 1996-2005 by Solar Designer
 */

/*
 * Some global parameters.
```

```
 */

#ifndef _JOHN_PARAMS_H
#define _JOHN_PARAMS_H

#include <limits.h>

/*
 * John's version number.
 */
#define JOHN_VERSION              "1.6.38"
#define JOHN_BOINC_VERSION        "1.0"
/*
 *  Is  this  a  system-wide  installation?  *BSD  ports  and  Linux
distributions
 * will probably want to set this to 1 for their builds of John.
 */
#ifndef JOHN_SYSTEMWIDE
#define JOHN_SYSTEMWIDE                0
#endif

#if JOHN_SYSTEMWIDE
#define JOHN_SYSTEMWIDE_EXEC      "/usr/libexec/john"
#define JOHN_SYSTEMWIDE_HOME      "/usr/share/john"
#define JOHN_PRIVATE_HOME         "~/.john"
#endif

/*
 * Crash recovery file format version strings.
 */
#define RECOVERY_VERSION_0        "REC0"
#define RECOVERY_VERSION_1        "REC1"
#define RECOVERY_VERSION_2        "REC2"
#define RECOVERY_VERSION_CURRENT  RECOVERY_VERSION_2

/*
 * Charset file format version string.
 */
#define CHARSET_VERSION                "CHR1"

/*
 * Timer interval in seconds.
 */
#define TIMER_INTERVAL            1

/*
 * Default crash recovery file saving delay in timer intervals.
 */
#define TIMER_SAVE_DELAY          (600 / TIMER_INTERVAL)

/*
 * Benchmark time in seconds, per cracking algorithm.
 */
#define BENCHMARK_TIME            5
```

```
/*
 * Number of salts to assume when benchmarking.
 */
#define BENCHMARK_MANY            0x100

/*
 * File names.
 */
#define CFG_FULL_NAME             "$JOHN/pwd_ldr.conf"
#define CFG_ALT_NAME              "$JOHN/pwd_ldr.ini"
#if JOHN_SYSTEMWIDE
#define CFG_PRIVATE_FULL_NAME           JOHN_PRIVATE_HOME
"/john.conf"
#define CFG_PRIVATE_ALT_NAME      JOHN_PRIVATE_HOME "/john.ini"
#define POT_NAME             JOHN_PRIVATE_HOME "/john.pot"
#define LOG_NAME             JOHN_PRIVATE_HOME "/john.log"
#define RECOVERY_NAME             JOHN_PRIVATE_HOME "/john.rec"
#else
#define POT_NAME             "$JOHN/pwd_ldr.pot"
#define LOG_NAME             "$JOHN/pwd_ldr.log"
#define RECOVERY_NAME             "$JOHN/pwd_ldr.rec"
#endif
#define LOG_SUFFIX                ."log"
#define RECOVERY_SUFFIX                ."rec"
#define WORDLIST_NAME             "$JOHN/password.lst"

//BOINC Add parameters
#define DB_NAME                   "john_db"
#define WU_TEMPLATE               "john_wu.xml"
#define RE_TEMPLATE               "john_re.xml"
#define DEFAULT_PRIORITY    20
#define DEF_APP_NAME        "john_boinc"
#define BATCH_NOT_SET       -1
#define DEFAULT_INC_WU_SIZE  1000000
//Batch flags
#define SINGLE_MODE               1
#define WORDLIST_MODE       2
#define INCREMENTAL_MODE    3
#define BATCH_COMPLETE      4
#define BATCH_ERROR             -1
//Status Flags
#define NOT_WORKED          1
#define WORKING             2
#define CRACKED             3
#define ERROR               -1
/*
 * Configuration file section names.
 */
#define SECTION_OPTIONS                "Options"
#define SECTION_RULES             "List.Rules:"
#define SUBSECTION_SINGLE         "Single"
#define SUBSECTION_WORDLIST       "Wordlist"
#define SECTION_INC               "Incremental:"
```

```
#define SECTION_EXT               "List.External:"
#define SECTION_BOINC             "Boinc"
/*
 *  Hash  table  sizes.  These  are  also  hardcoded  into  the  hash
functions.
 */
#define SALT_HASH_SIZE            0x400
#define PASSWORD_HASH_SIZE_0      0x10
#define PASSWORD_HASH_SIZE_1      0x100
#define PASSWORD_HASH_SIZE_2      0x1000

/*
 * Password hash table thresholds. These are the counts of entries
required
 * to enable the corresponding hash table size.
 */
#define PASSWORD_HASH_THRESHOLD_0 (PASSWORD_HASH_SIZE_0 / 2)
#define PASSWORD_HASH_THRESHOLD_1 (PASSWORD_HASH_SIZE_1 / 4)
#define PASSWORD_HASH_THRESHOLD_2 (PASSWORD_HASH_SIZE_2 / 4)

/*
 * Tables of the above values.
 */
extern int password_hash_sizes[3];
extern int password_hash_thresholds[3];

/*
 * Cracked password hash size, used while loading.
 */
#define CRACKED_HASH_LOG          10
#define CRACKED_HASH_SIZE         (1 << CRACKED_HASH_LOG)

/*
 * Password hash function to use while loading.
 */
#define LDR_HASH_SIZE  (PASSWORD_HASH_SIZE_2     *     sizeof(struct
db_password *))
#define LDR_HASH_FUNC  (format->methods.binary_hash[2])

/*
 * Buffered keys hash size, used for "single crack" mode.
 */
#define SINGLE_HASH_LOG                5
#define SINGLE_HASH_SIZE          (1 << SINGLE_HASH_LOG)

/*
 *  Minimum  buffered  keys  hash  size,  used  if  min_keys_per_crypt  is
even less.
 */
#define SINGLE_HASH_MIN                8

/*
 * Shadow file entry table hash size, used by unshadow.
 */
```

```
#define SHADOW_HASH_LOG                 8
#define SHADOW_HASH_SIZE            (1 << SHADOW_HASH_LOG)

/*
 * Hash and buffer sizes for unique.
 */
#define UNIQUE_HASH_LOG                 17
#define UNIQUE_HASH_SIZE            (1 << UNIQUE_HASH_LOG)
#define UNIQUE_BUFFER_SIZE          0x800000

/*
 * Maximum number of GECOS words per password to load.
 */
#define LDR_WORDS_MAX               0x10

/*
 * Maximum number of GECOS words to try in pairs.
 */
#define SINGLE_WORDS_PAIR_MAX           4

/*
 * Charset parameters.
 * Be careful if you change these, ((SIZE ** LENGTH) * SCALE) should
fit
 * into 64 bits. You can reduce the SCALE if required.
 */
#define CHARSET_MIN                 ' '
#define CHARSET_MAX                 0x7E
#define CHARSET_SIZE                (CHARSET_MAX - CHARSET_MIN + 1)
#define CHARSET_LENGTH              8
#define CHARSET_SCALE               0x100

/*
 * Compiler parameters.
 */
#define C_TOKEN_SIZE                0x100
#define C_UNGET_SIZE                (C_TOKEN_SIZE + 4)
#define C_EXPR_SIZE                 0x100
#define C_STACK_SIZE                ((C_EXPR_SIZE + 4) * 4)
#define C_ARRAY_SIZE                0x1000000
#define C_DATA_SIZE                 0x8000000

/*
 * Buffer size for rules.
 */
#define RULE_BUFFER_SIZE            0x100

/*
 * Maximum number of character ranges for rules.
 */
#define RULE_RANGES_MAX                 8

/*
```

```
 * Buffer size for words while applying rules, should be at least as
large
 * as PLAINTEXT_BUFFER_SIZE.
 */
#define RULE_WORD_SIZE            0x80

/*
 * Buffer size for plaintext passwords.
 */
#define PLAINTEXT_BUFFER_SIZE          0x80

/*
 * Buffer size for fgets().
 */
#define LINE_BUFFER_SIZE          0x400

/*
 * john.pot and log file buffer sizes, can be zero.
 */
#define POT_BUFFER_SIZE                0x1000
#define LOG_BUFFER_SIZE                0x1000

/*
 * Buffer size for path names.
 */
#ifdef PATH_MAX
#define PATH_BUFFER_SIZE          PATH_MAX
#else
#define PATH_BUFFER_SIZE          0x400
#endif

#endif
```

O. SINGLE.H

```
/*
 * This file is part of John the Ripper password cracker,
 * Copyright (c) 1996-98 by Solar Designer
 */

/*
 * "Single crack" mode.
 */

#ifndef _JOHN_SINGLE_H
#define _JOHN_SINGLE_H

#include "loader.h"
//#include "boinc.h"
/*
 * Runs the cracker.
 */
extern void do_single_crack(struct db_main *db, struct BATCH *b);
```

```
#endif
```

P. SINGLE.C

```
/*
 * This file is part of John the Ripper password cracker,
 * Copyright (c) 1996-99,2003,2004 by Solar Designer
 */

#include <stdio.h>
#include <string.h>

#include "misc.h"
#include "params.h"
#include "memory.h"
#include "signals.h"
#include "loader.h"
#include "logger.h"
#include "status.h"
#include "recovery.h"
#include "rpp.h"
#include "rules.h"
#include "external.h"
#include "cracker.h"
#include "boinc.h"

//static int progress = 0;
//static int rec_rule;

//static struct db_main *single_db;
//static int rule_number, rule_count;
//static int length, key_count;
//static struct db_keys *guessed_keys;
//static struct rpp_context *rule_ctx;
/*
static void save_state(FILE *file)
{
	fprintf(file, "%d\n", rec_rule);
}

static int restore_rule_number(void)
{
	if (rule_ctx)
	for (rule_number = 0; rule_number < rec_rule; rule_number++)
	if (!rpp_next(rule_ctx)) return 1;

	return 0;
}

static int restore_state(FILE *file)
{
	if (fscanf(file, "%d\n", &rec_rule) != 1) return 1;
```

```
	return restore_rule_number();
}

static int get_progress(void)
{
	if (progress) return progress;

	return rule_number * 100 / (rule_count + 1);
}

static void single_alloc_keys(struct db_keys **keys)
{
	int hash_size = sizeof(struct db_keys_hash) +
		sizeof(struct db_keys_hash_entry) * (key_count - 1);

	if (!*keys) {
		*keys = mem_alloc_tiny(
			sizeof(struct db_keys) - 1 + length * key_count,
			MEM_ALIGN_WORD);
		(*keys)->hash = mem_alloc_tiny(hash_size,
MEM_ALIGN_WORD);
	}

	(*keys)->count = 0;
	(*keys)->ptr = (*keys)->buffer;
	(*keys)->rule = rule_number;
	(*keys)->lock = 0;
	memset((*keys)->hash, -1, hash_size);
}

static void single_init(void)
{
	struct db_salt *salt;

	log_event("Proceeding with \"single crack\" mode");

	progress = 0;

	length = single_db->format->params.plaintext_length;
	key_count = single_db->format->params.min_keys_per_crypt;
	if (key_count < SINGLE_HASH_MIN) key_count = SINGLE_HASH_MIN;

	if (rpp_init(rule_ctx, SUBSECTION_SINGLE)) {
		log_event("! No \"single crack\" mode rules found");
		fprintf(stderr, "No \"single crack\" mode rules found in
%s\n",
			cfg_name);
		error();
	}

	rules_init(length);
	rec_rule = rule_number = 0;
	rule_count = rules_count(rule_ctx, 0);
```

```
	log_event("-	%d	preprocessed	word	mangling	rules",
rule_count);

	status_init(get_progress, 0);

	rec_restore_mode(restore_state);
	rec_init(single_db, save_state);

	salt = single_db->salts;
	do {
		single_alloc_keys(&salt->keys);
	} while ((salt = salt->next));

	if (key_count > 1)
	log_event("-	Allocated	%d	buffer%s	of	%d	candidate
passwords%s",
		single_db->salt_count,
		single_db->salt_count != 1 ? "s" : "",
		key_count,
		single_db->salt_count != 1 ? " each" : "");

	guessed_keys = NULL;
	single_alloc_keys(&guessed_keys);

	crk_init(single_db, NULL, guessed_keys);
}

static int single_key_hash(char *key)
{
	int pos, hash = 0;

	for (pos = 0; pos < length && *key; pos++) {
		hash <<= 1;
		hash ^= *key++;
	}

	hash ^= hash >> SINGLE_HASH_LOG;
	hash ^= hash >> (2 * SINGLE_HASH_LOG);
	hash &= SINGLE_HASH_SIZE - 1;

	return hash;
}

static int single_add_key(struct db_keys *keys, char *key)
{
	int index, hash;
	struct db_keys_hash_entry *entry;

	if ((index = keys->hash->hash[single_key_hash(key)]) >= 0)
	do {
		entry = &keys->hash->list[index];
		if (!strncmp(key, &keys->buffer[entry->offset], length))
			return 0;
```

```
	} while ((index = entry->next) >= 0);

	index = keys->hash->hash[hash = single_key_hash(keys->ptr)];
	if (index == keys->count)
		keys->hash->hash[hash] = keys->hash->list[index].next;
	else
	if (index >= 0) {
		entry = &keys->hash->list[index];
		while ((index = entry->next) >= 0) {
			if (index == keys->count) {
				entry->next = keys->hash->list[index].next;
				break;
			}
			entry = &keys->hash->list[index];
		}
	}

	index = keys->hash->hash[hash = single_key_hash(key)];
	entry = &keys->hash->list[keys->count];
	entry->next = index;
	entry->offset = keys->ptr - keys->buffer;
	keys->hash->hash[hash] = keys->count;

	strnfcpy(keys->ptr, key, length);
	keys->ptr += length;

	return ++(keys->count) >= key_count;
}

static int single_process_buffer(struct db_salt *salt)
{
	struct db_salt *current;
	struct db_keys *keys;
	size_t size;

	if (crk_process_salt(salt)) return 1;

	keys = salt->keys;
	keys->count = 0;
	keys->ptr = keys->buffer;
	keys->lock++;

	if (guessed_keys->count) {
		keys = mem_alloc(size = sizeof(struct db_keys) - 1 +
			length * guessed_keys->count);
		memcpy(keys, guessed_keys, size);

		keys->ptr = keys->buffer;
		do {
			current = single_db->salts;
			do {
				if (current == salt) continue;
				if (!current->list) continue;
```

```
                        if (single_add_key(current->keys, keys->ptr))
                        if (single_process_buffer(current)) return 1;
                  } while ((current = current->next));
                  keys->ptr += length;
            } while (--keys->count);

            MEM_FREE(keys);
      }

      keys = salt->keys;
      keys->lock--;
      if (!keys->count && !keys->lock) keys->rule = rule_number;

      return 0;
}

static   int   single_process_pw(struct   db_salt   *salt,   struct
db_password *pw,
      char *rule)
{
      struct db_keys *keys;
      struct list_entry *first, *second;
      int first_number, second_number;
      char pair[RULE_WORD_SIZE];
      int split;
      char *key;
      unsigned int c;

      keys = salt->keys;

      first_number = 0;
      if ((first = pw->words->head))
      do {
            if ((key = rules_apply(first->data, rule, 0)))
            if (ext_filter(key))
            if (single_add_key(keys, key))
            if (single_process_buffer(salt)) return 1;
            if (!salt->list) return 0;

            if (++first_number > SINGLE_WORDS_PAIR_MAX) continue;

            c = (unsigned int)first->data[0] | 0x20;
            if (c < 'a' || c > 'z') continue;

            second_number = 0;
            second = pw->words->head;

            do
            if (first != second) {
                  if ((split = strlen(first->data)) < length) {
                        strnzcpy(pair, first->data, RULE_WORD_SIZE);
                        strnzcat(pair, second->data, RULE_WORD_SIZE);

                        if ((key = rules_apply(pair, rule, split)))
```

```
                        if (ext_filter(key))
                        if (single_add_key(keys, key))
                        if (single_process_buffer(salt)) return 1;
                        if (!salt->list) return 0;
                  }

                  if (first->data[1]) {
                        pair[0] = first->data[0];
                        pair[1] = 0;
                        strnzcat(pair, second->data, RULE_WORD_SIZE);

                        if ((key = rules_apply(pair, rule, 1)))
                        if (ext_filter(key))
                        if (single_add_key(keys, key))
                        if (single_process_buffer(salt)) return 1;
                        if (!salt->list) return 0;
                  }
            } while (++second_number <= SINGLE_WORDS_PAIR_MAX &&
                  (second = second->next));
      } while ((first = first->next));

      return 0;
}

static int single_process_salt(struct db_salt *salt, char *rule)
{
      struct db_keys *keys;
      struct db_password *pw;

      keys = salt->keys;

      pw = salt->list;
      do {
            if (single_process_pw(salt, pw, rule)) return 1;
            if (!salt->list) return 0;
      } while ((pw = pw->next));

      if (keys->count && rule_number - keys->rule > (key_count <<
1))
            if (single_process_buffer(salt)) return 1;

      if (!keys->count) keys->rule = rule_number;

      return 0;
}

static void single_run(void)
{
      char *prerule, *rule;
      struct db_salt *salt;
      int min, saved_min;

      saved_min = rec_rule;
      while ((prerule = rpp_next(rule_ctx))) {
```

```
            if (!(rule = rules_reject(prerule, single_db))) {
                  log_event("- Rule #%d: '%.100s' rejected",
                        ++rule_number, prerule);
                  continue;
            }

            if (strcmp(prerule, rule))
                  log_event("- Rule #%d: '%.100s' accepted as '%s'",
                        rule_number + 1, prerule, rule);
            else
                  log_event("- Rule #%d: '%.100s' accepted",
                        rule_number + 1, prerule);

            if (saved_min != rec_rule) {
                  log_event("- Oldest still in use is now rule #%d",
                        rec_rule + 1);
                  saved_min = rec_rule;
            }

            min = rule_number;

            salt = single_db->salts;
            do {
                  if (!salt->list) continue;
                  if (single_process_salt(salt, rule)) return;
                  if (salt->keys->rule < min) min = salt->keys->rule;
            } while ((salt = salt->next));

            rec_rule = min;
            rule_number++;
      }
}

static void single_done(void)
{
      struct db_salt *salt;

      if (!event_abort) {
            log_event("- Processing the remaining buffered "
                  "candidate passwords");

            if ((salt = single_db->salts))
            do {
                  if (!salt->list) continue;
                  if (salt->keys->count)
                  if (single_process_buffer(salt)) break;
            } while ((salt = salt->next));

            progress = 100;
      }

      rec_done(event_abort);
}
*/
```

```
void do_single_crack(struct db_main *db, struct BATCH *b)
{
        if(gen_work(b)){
            fprintf(stderr,"unable  to  generate  work  for  single
mode\n");
         b->mode_flag=BATCH_ERROR;
         return;
        }
        b->mode_flag++;
        fprintf(stderr,"Mode flag after gen_work in single =%d\n"
                ,b->mode_flag);

}
```

Q. WORDLIST.H

```
/*
 * This file is part of John the Ripper password cracker,
 * Copyright (c) 1996-98 by Solar Designer
 */

/*
 * Wordlist cracker.
 */

#ifndef _JOHN_WORDLIST_H
#define _JOHN_WORDLIST_H

#include "loader.h"
//#include "boinc.h"
/*
 * Runs the wordlist cracker reading words from the supplied file
name, or
 * stdin if name is NULL.
 */
extern void do_wordlist_crack(struct db_main *db, struct BATCH *b,
int rules);

#endif
```

R. WORDLIST.C

```
/*Distributed John - Work Generator - BOINC

 * This file is part of John the Ripper password cracker,
 * Copyright (c) 1996-99,2003,2004 by Solar Designer
 */

#include <stdio.h>
#include <sys/stat.h>
#include <unistd.h>
#include <string.h>
```

```
#include "misc.h"
#include "math.h"
#include "params.h"
#include "path.h"
#include "signals.h"
#include "loader.h"
#include "logger.h"
#include "status.h"
#include "recovery.h"
#include "rpp.h"
#include "rules.h"
#include "external.h"
#include "cracker.h"
#include "boinc.h"

static FILE *word_file = NULL;
static int progress = 0;

static int rec_rule;
//static long rec_pos;

static int rule_number, rule_count, line_number;
static int length;
static struct rpp_context *rule_ctx;
/*
static void save_state(FILE *file)
{
	fprintf(file, "%d\n%ld\n", rec_rule, rec_pos);
}
*/
static int restore_rule_number(void)
{
	if (rule_ctx)
	for (rule_number = 0; rule_number < rec_rule; rule_number++)
	if (!rpp_next(rule_ctx)) return 1;

	return 0;
}
/*
static void restore_line_number(void)
{
	char line[LINE_BUFFER_SIZE];

	for (line_number = 0; line_number < rec_pos; line_number++)
	if (!fgets(line, sizeof(line), word_file)) {
		if (ferror(word_file))
			pexit("fgets");
		else {
			fprintf(stderr, "fgets: Unexpected EOF\n");
			error();
		}
	}
}
```

```
static int restore_state(FILE *file)
{
      if  (fscanf(file,  "%d\n%ld\n",  &rec_rule,  &rec_pos)  !=  2)
return 1;

      if (restore_rule_number()) return 1;

      if (word_file == stdin)
            restore_line_number();
      else
            if (fseek(word_file, rec_pos, SEEK_SET)) pexit("fseek");

      return 0;
}

static void fix_state(void)
{
      rec_rule = rule_number;

      if (word_file == stdin)
            rec_pos = line_number;
      else
      if ((rec_pos = ftell(word_file)) < 0) {
#ifdef __DJGPP__
            if (rec_pos != -1)
                  rec_pos = 0;
            else
#endif
                  pexit("ftell");
      }
}
*/
static int get_progress(void)
{
      struct stat file_stat;
      long pos;
      int64 x100;

      if (!word_file) return progress;

      if (word_file == stdin) return -1;

      if (fstat(fileno(word_file), &file_stat)) pexit("fstat");

      if ((pos = ftell(word_file)) < 0) {
#ifdef __DJGPP__
            if (pos != -1)
                  pos = 0;
            else
#endif
                  pexit("ftell");
      }
```

```
      mul32by32(&x100, pos, 100);
      return
            (rule_number * 100 +
            div64by32lo(&x100, file_stat.st_size + 1)) / rule_count;
}

static char *dummy_rules_apply(char *word, char *rule, int split)
{
      word[length] = 0;

      return word;
}

void  do_wordlist_crack(struct  db_main  *db,  struct  BATCH  *b,  int
rules)
{
//    char line[LINE_BUFFER_SIZE];
      struct rpp_context ctx;
      char *prerule, *rule;//, *word;
//    char last[RULE_WORD_SIZE];
      char *(*apply)(char *word, char *rule, int split);
    int generated_work;
 //   char *format;
      fprintf(stderr,"Inside do wordlist_crack\n");
/*
      if (name) {
            if (!(word_file = fopen(path_expand(name), "r")))
                  pexit("fopen: %s", path_expand(name));
            log_event("- Wordlist file: %.100s", path_expand(name));
      } else {
            word_file = stdin;
            log_event("- Reading candidate passwords from stdin");
      }

    batch_load_format(b,&format);
    fprintf(stderr,"format= %s\n",format);

   if ((db->format = fmt_list))
   {
        fprintf(stderr,"After form assignment\n");
        do{
            if(!strcmp(format,db->format->params.format_name)){
                fprintf(stderr,"Inside form loop %d\n"
                        ,db->format->params.plaintext_length);
                fmt_init(db->format);
                break;
            }
        } while ((db->format = db->format->next));
   }
    fprintf(stderr,"After form loop\n");
*/
      length = db->format->params.plaintext_length;
    fprintf(stderr,"After Length=%d in do_wordlist_crack\n",length);
      if (rules) {
```

```
            if (rpp_init(rule_ctx = &ctx, SUBSECTION_WORDLIST)) {
                  log_event("! No wordlist mode rules found");
                  fprintf(stderr, "No wordlist mode rules found in
%s\n",
                        cfg_name);
                  error();
            }

            rules_init(length);
            rule_count = rules_count(&ctx, -1);

            log_event("- %d preprocessed word mangling rules",
rule_count);

            apply = rules_apply;
      } else {
            rule_ctx = NULL;
            rule_count = 1;

            log_event("- No word mangling rules");

            apply = dummy_rules_apply;
      }

      generated_work = line_number = rule_number = 0;
    rec_rule = b->s_loc;
      status_init(get_progress, 0);
    if (restore_rule_number()){
       fprintf(stderr,"End of Rules\n");
       b->mode_flag++;
       b->s_loc=0;
    }
/*    rec_restore_mode(restore_state);
      rec_init(db, save_state);

      crk_init(db, fix_state, NULL);
*/
      if (rules) prerule = rpp_next(&ctx); else prerule = "";
      rule = "";
/*
      memset(last, ' ', length + 1);
      last[length + 2] = 0;
      fprintf(stderr,"before if (prerule)\n");
*/    if (prerule)
      do {
            if (rules) {
                  if ((rule = rules_reject(prerule, db))) {
                        if (strcmp(prerule, rule))
                              log_event("- Rule #%d: '%.100s'"
                                    " accepted as '%.100s'",
                                    rule_number + 1, prerule, rule);
                        else
                              log_event("- Rule #%d: '%.100s'"
                                    " accepted",
```

```
                                    rule_number + 1, prerule);
                        } else
                              log_event("- Rule #%d: '%.100s' rejected",
                                    rule_number + 1, prerule);
                  }
                  //rule was accepted
                  if (rule){
                    if(gen_work(b)){
                          fprintf(stderr,"Failed to generate work\n");
                        b->mode_flag=BATCH_ERROR;
                        break;
                   }else
                       generated_work=1;
                }
                  if (rules) {
                        if (!(rule = rpp_next(&ctx))){
                       b->mode_flag++;
                       b->s_loc=0;
                       break;
                   }
                        rule_number++;
                   b->s_loc=rule_number;
                   if (generated_work)
                       rules=0;
//                      fprintf(stderr,"rule_numer:%d\n",rule_number);
//                      line_number = 0;
//                      if (fseek(word_file, 0, SEEK_SET)) pexit("fseek");
                  }
            } while (rules);


/*    crk_done();

            rec_done(event_abort);

            if (ferror(word_file)) pexit("fgets");

            if (name) {
                  if (event_abort)
                        progress = get_progress();
                  else
                        progress = 100;

                  if (fclose(word_file)) pexit("fclose");
                  word_file = NULL;
            }*/
}
```

THIS PAGE INTENTIONALLY LEFT BLANK

APPENDIX B: PASSWORD LOADER CODE

A. JOHHN.C

```
/*************************************************************
 *  Distributed BOINC Password Cracker- Using John The Ripper
 *
 *  This is a BOINC app that cracks passwords. This part of the
 *  application is the work generator.  All BOINC projects require
 *  some kind of work generator for BOINC to produce result units
 *  to send to clients.
 *
 *  7 June 2005 to use newest BOINC API as of vers. 4.52
 *
 *  John Crumpacker <jrcrumpa@nps.edu>  -  7 June 2005
 *  Department of Computer Science,
 *  Naval Postgraduate School, Monterey, California
 *  @(#) $Version: 1.00$

*************************************************************
***/
/*
 * This file is part of John the Ripper password cracker,
 * Copyright (c) 1996-2004 by Solar Designer
 * Modified for BOINC by John Crumpacker
 * This is the work generator version of John the Ripper its only
 * purpose is to create work units for client computers.
 */


//Functions added to make this distributed
#include "boinc.h"


// C lib
#include <stdio.h>
#include <unistd.h>
#include <string.h>
#include <stdlib.h>
#include <sys/stat.h>

// John the ripper includes
#include "arch.h"
#include "misc.h"
#include "params.h"
#include "path.h"
#include "memory.h"
#include "list.h"
#include "tty.h"
#include "signals.h"
#include "common.h"
#include "formats.h"
```

```
#include "loader.h"
#include "logger.h"
#include "status.h"
#include "options.h"
#include "john_config.h"
#include "bench.h"
#include "charset.h"
#include "single.h"
#include "wordlist.h"
#include "inc.h"
#include "external.h"
#include "batch.h"

#if CPU_DETECT
extern int CPU_detect(void);
#endif

extern struct fmt_main fmt_DES, fmt_BSDI, fmt_MD5, fmt_BF;
extern struct fmt_main fmt_AFS, fmt_LM;

extern int unshadow(int argc, char **argv);
extern int unafs(int argc, char **argv);
extern int unique(int argc, char **argv);

static struct db_main database;
static struct fmt_main dummy_format;

static void john_register_one(struct fmt_main *format)
{
      if (options.format)
      if (strcmp(options.format, format->params.label)) return;

      fmt_register(format);
}

static void john_register_all(void)
{
      if (options.format) strlwr(options.format);

      john_register_one(&fmt_DES);
      john_register_one(&fmt_BSDI);
      john_register_one(&fmt_MD5);
      john_register_one(&fmt_BF);
      john_register_one(&fmt_AFS);
      john_register_one(&fmt_LM);

      if (!fmt_list) {
            fprintf(stderr,   "Unknown   ciphertext   format   name
requested\n");
            error();
      }
}

static void john_log_format(void)
```

```
{
      int min_chunk, chunk;

      log_event("- Hash type: %.100s (lengths up to %d%s)",
            database.format->params.format_name,
            database.format->params.plaintext_length,
            database.format->methods.split != fmt_default_split ?
            ", longer passwords split" : "");

      log_event("- Algorithm: %.100s",
            database.format->params.algorithm_name);

      chunk       =       min_chunk       =       database.format-
>params.max_keys_per_crypt;
      if (options.flags & (FLG_SINGLE_CHK | FLG_BATCH_CHK) &&
          chunk < SINGLE_HASH_MIN)
                chunk = SINGLE_HASH_MIN;
      if (chunk > 1)
            log_event("- Candidate passwords %s be buffered and "
                  "tried in chunks of %d",
                  min_chunk > 1 ? "will" : "may",
                  chunk);
}

static char *john_loaded_counts(void)
{
      static char s_loaded_counts[80];

      if (database.password_count == 1)
            return "1 password hash";

      sprintf(s_loaded_counts,
            database.salt_count > 1 ?
            "%d password hashes with %d different salts" :
            "%d password hashes with no different salts",
            database.password_count,
            database.salt_count);

      return s_loaded_counts;
}


static void john_load(void)
{
      struct list_entry *current;
      umask(077);
      fprintf(stderr,"Entering john_load\n");
/*    if (options.flags & FLG_EXTERNAL_CHK)
            ext_init(options.external);

      if (options.flags & FLG_MAKECHR_CHK) {
            options.loader.flags |= DB_CRACKED;
            ldr_init_database(&database, &options.loader);
```

```
            if (options.flags & FLG_PASSWD) {
                  ldr_show_pot_file(&database, POT_NAME);

                  database.options->flags |= DB_PLAINTEXTS;
                  if ((current = options.passwd->head))
                  do {
                        ldr_show_pw_file(&database, current->data);
                  } while ((current = current->next));
            } else {
                  database.options->flags |= DB_PLAINTEXTS;
                  ldr_show_pot_file(&database, POT_NAME);
            }

            return;
      }
//does not need this...
      if (options.flags & FLG_STDOUT) {
            ldr_init_database(&database, &options.loader);
            database.format = &dummy_format;
            memset(&dummy_format, 0, sizeof(dummy_format));
            dummy_format.params.plaintext_length = options.length;
            dummy_format.params.flags = FMT_CASE | FMT_8_BIT;
      }
*/
      if (options.flags & FLG_APP) {
            fprintf(stderr,"app_name:%s\n",options.app_name);
            insert_app_name(options.app_name);
      }else{
            insert_app_name(DEF_APP_NAME);
      }
      if (options.flags & FLG_PASSWD) {
            /*won't need this... Going to have a different interface
                  to deal with this stuff. Cracked Passwords will be
                  in a MySQL DB*/
/*          if (options.flags & FLG_SHOW_CHK) {
                  options.loader.flags |= DB_CRACKED;
                  ldr_init_database(&database, &options.loader);

                  ldr_show_pot_file(&database, POT_NAME);

                  if ((current = options.passwd->head))
                  do {
                        ldr_show_pw_file(&database, current->data);
                  } while ((current = current->next));

                  printf("%s%d password hash%s cracked, %d left\n",
                        database.guess_count ? "\n" : "",
                        database.guess_count,
                        database.guess_count != 1 ? "es" : "",
                        database.password_count -
                        database.guess_count);

                  return;
            }*/
```

```
            fprintf(stderr,"Inside john_load\n");
            if (options.flags & (FLG_SINGLE_CHK | FLG_BATCH_CHK))
                  options.loader.flags |= DB_WORDS;
/*          else
            if (mem_saving_level)
                  options.loader.flags &= ~DB_LOGIN;*/
            ldr_init_database(&database, &options.loader);

            if ((current = options.passwd->head))
            do {
                  ldr_load_pw_file(&database, current->data);
            } while ((current = current->next));

            if ((options.flags & FLG_CRACKING_CHK) &&
                database.password_count) {
                  log_init(LOG_NAME, NULL, options.session);
                  if (status_restored_time)
                        log_event("Continuing      an      interrupted
session");
                  else
                        log_event("Starting a new session");
                  log_event("Loaded      a      total      of      %s",
john_loaded_counts());
            }
//Do not need a pot file... in work gen... maybe in Assimilator...
/*          ldr_load_pot_file(&database, POT_NAME);
//Do not need to fix the DB... because not stored locally
//          ldr_fix_database(&database);
//will change this to log PASSWD_ITEMS loaded...
            if (database.password_count) {
                  log_event("Remaining %s", john_loaded_counts());
                  printf("Loaded %s (%s [%s])\n",
                        john_loaded_counts(),
                        database.format->params.format_name,
                        database.format->params.algorithm_name);
            } else {
                  log_discard();
                  puts("No password hashes loaded");
            }

            if  ((options.flags  &  FLG_PWD_REQ)  &&  !database.salts)
exit(0);*/
      }
      fprintf(stderr,"exiting john_load\n");
}

static void john_init(int argc, char **argv)
{
#if CPU_DETECT
      if (!CPU_detect()) {
#if CPU_REQ
#if CPU_FALLBACK
#if defined(__DJGPP__) || defined(__CYGWIN32__)
```

```
#error CPU_FALLBACK is incompatible with the current DOS and Win32
code
#endif
            execv(JOHN_SYSTEMWIDE_EXEC    "/"    CPU_FALLBACK_BINARY,
argv);
            perror("execv");
#endif
            fprintf(stderr, "Sorry, %s is required\n", CPU_NAME);
            error();
#endif
      }
#endif
      fprintf(stderr, "Inside John_init\n");
//Should be able to leave this be...
      path_init(argv);

//In params.h JOHN_SYSTEMWIDE is set to 0.. not enabled
#if JOHN_SYSTEMWIDE
      cfg_init(CFG_PRIVATE_FULL_NAME, 1);
      cfg_init(CFG_PRIVATE_ALT_NAME, 1);
#endif
/*Need this for sure... the difference being that it accepts both
john.conf   and   john.ini   files...      Add   template   names   to
john.conf/.ini or
just take commandline. If I modify this can have a default and have
flexiability
for change later on. */
      cfg_init(CFG_FULL_NAME, 1);
      cfg_init(CFG_ALT_NAME, 0);

/*this will create the connection
to the BOINC DB and bring in data from MySQL DB.*/
      boinc_DB_init();



      status_init(NULL, 1);
/* Added Application name to the command line options and removed
all other
 *  functionality.  To  ensure  this  was  only  a  work  generator.  Used
john logging
 * functionality for error reporting.*/
      opt_init(argc, argv);
/*Load all formats*/
      john_register_all();
      common_init();

      sig_init();
//only need parts of this ...
      john_load();
      fprintf(stderr,"Exit of John_init\n");

}
```

```
static void john_done(void)
{
      path_done();

      if ((options.flags & FLG_CRACKING_CHK) &&
          !(options.flags & FLG_STDOUT)) {
            if (event_abort)
                  log_event("Session aborted");
            else
                  log_event("Session completed");
      }
      log_done();

      check_abort(0);
}

int main(int argc, char **argv)
{
      char *name;

#ifdef __DJGPP__
      if (--argc <= 0) return 1;
      if ((name = strrchr(argv[0], '/')))
            strcpy(name + 1, argv[1]);
      name = argv[1];
      argv[1] = argv[0];
      argv++;
#else
      if (!argv[0])
            name = "";
      else
      if ((name = strrchr(argv[0], '/')))
            name++;
      else
            name = argv[0];
#endif

#ifdef __CYGWIN32__
      if (strlen(name) > 4)
      if (!strcmp(strlwr(name) + strlen(name) - 4, .”exe"))
            name[strlen(name) - 4] = 0;
#endif

      if (!strcmp(name, "unshadow"))
            return unshadow(argc, argv);

      if (!strcmp(name, "unafs"))
            return unafs(argc, argv);

      if (!strcmp(name, "unique"))
            return unique(argc, argv);

      john_init(argc, argv);
//    john_run();
```

```
      john_done();

      return 0;
}
```

B. BOINC.H

```
#ifndef _JOHN_BOINC_H
#define _JOHN_BOINC_H

// #include <boinc_db.h>
// #include "john_db.h"

#ifdef __cplusplus
extern "C" {
#endif

/*This function initializes connections to both
the BOINC db and John db*/
void boinc_DB_init();

/*Loads passwords from file into john_db*/
void boinc_pw_load(char *login, char *ciphertext, char *gecos,char
*home,
            char *UID, char *GID, char *shell,char *format);

/*Checks for duplicate passwords (both login and ciphertext match
another password in the list) returns true if no duplicate found.*/
int pw_check_dup(char *login, char *ciphertext, char *format);

//int flag_check(int flag);

int gen_work(int flag,int num_rules);

void insert_app_name(char *);

void john_sleep();

int daemons_stopped();

int check_ready_wu();

void batch_assign();

struct BATCH{
    int id;
    int mode_flag;
    int batch_priority;
    int s_loc;//s=start ... Entry or Rule #
    int s_length;
    int s_fixed;
    int s_count;
```

```
    int s_num0;
    int s_num1;
    int s_num2;
    int s_num3;
    int s_num4;
    int s_num5;
    int s_num6;
    int s_num7;
    int f_loc;
    int f_length;
    int f_fixed;
    int f_count;
    int f_num0;
    int f_num1;
    int f_num2;
    int f_num3;
    int f_num4;
    int f_num5;
    int f_num6;
    int f_num7;

//    void clear();
};
//BATCH& operator=(const BATCH &source);
//void b_clear(BATCH &b);
#ifdef __cplusplus
}
#endif
#endif
```

C. BOINC.C

```
#include <vector>
#include <algorithm>
#include <stdlib.h>
#include <stdio.h>
#include <string.h>
#include <unistd.h>
//BOINC Backend Libraries
#include "filesys.h"
#include "backend_lib.h"
#include "util.h"
#include "sched_config.h"
#include "sched_util.h"
#include "sched_msgs.h"
#include "parse.h"
#include "crypt.h"
//#include "error_numbers.h"

#include "boinc.h"

//Distributed John Includes
extern "C" {
```

```
#include <sys/stat.h>
#include "misc.h"
#include "params.h"
#include "john_config.h"
#include "logger.h"

}

#include <boinc_db.h>
#include "john_db.h"

DB_APP app;
//R_RSA_PRIVATE_KEY key;
SCHED_CONFIG conf;
int num_ready_wu;
int wu_count = 0;
int trigger = 0;

//void b_clear(BATCH &b) {memset(b,0,sizeof(b));}
/*
BATCH& operator=(const BATCH &s){
   id=s.id;
   mode_flag=s.mode_flag;
   batch_priority=s.batch_priority;
   s_loc=s.s_loc;//s=start ... Entry or Rule #
   s_length=s.s_length;
   s_fixed=s.s_fixed;
   s_count=s.s_count;
   s_num0=s.s_num0;
   s_num1=s.s_num1;
   s_num2=s.s_num2;
   s_num3=s.s_num3;
   s_num4=s.s_num4;
   s_num5=s.s_num5;
   s_num6=s.s_num6;
   s_num7=s.s_num7;
   f_loc=s.f_loc;
   f_length=s.f_length;
   f_fixed=s.f_fixed;
   f_count=s.f_count;
   f_num0=s.f_num0;
   f_num1=s.f_num1;
   f_num2=s.f_num2;
   f_num3=s.f_num3;
   f_num4=s.f_num4;
   f_num5=s.f_num5;
   f_num6=s.f_num6;
   f_num7=s.f_num7;

}
*/
void boinc_DB_init(){
        int retval;
        int password_count = 0;
```

```
      char *DBname;
      char *Host_name;
      char *User_name;
      char *pwd;

        retval = conf.parse_file(.”.");
        if (retval){
                log_event("Can't parse config file\n");
                fprintf(stderr, "Can't parse BOINC config file\n");
                error();
        }
        retval    =    boinc_db.open(conf.db_name,    conf.db_host,
conf.db_user, conf.db_passwd);
        if (retval){
            fprintf(stderr,"Can not open BOINC db\n");
                error();
        }
      DBname = cfg_get_param(SECTION_BOINC,NULL,"DB_name");

      if (!DBname)
            DBname = DB_NAME;
      //need to add user to params... will not have a password would
have to
      //figure another way to save the password safely... Don't want
to deal
      //with that.
      Host_name = cfg_get_param(SECTION_BOINC,NULL,"Host_name");
      User_name = cfg_get_param(SECTION_BOINC,NULL,"User_name");
      pwd =cfg_get_param(SECTION_BOINC,NULL,"Pass_word");

//    fprintf(stderr,"DB_name    %s,    Host_name    %s,    User_name
%s\n",DBname,Host_name,User_name);

      retval = john_db.open(DBname,Host_name,User_name,pwd);
      if (retval){
            fprintf(stderr,"Can not open John DB\n");
            error();
      }
//    fprintf(stderr,"exit of boinc_DB_init\n");
      num_ready_wu = 0;
      num_ready_wu = cfg_get_int(SECTION_BOINC,NULL,"Max_wu");

      if (!(num_ready_wu) ){
            fprintf(stderr,"Failed to get Number of ready"
                        " workunits\n");
            error();
      }
}//END boinc_DB_init();

void john_sleep(){
      int retval;
      retval = sleep(6);
      if (retval){
```

```
            fprintf(stderr,"Sleep   interupted   by   amount:   %d\n",
retval);
      }
}

int daemons_stopped(){
      check_stop_daemons();
//    check_reread_trigger();
      return 0;
}

void insert_app_name(char *app_name){
      char buf[256];
      int retval;
      fprintf(stderr,"Inside app_name : %s\n",app_name);
      strcpy(app.name,app_name);
      fprintf(stderr,"after strncpy in app_name: %s\n",app.name);
      sprintf(buf,"where name='%s'",app.name);
      retval = app.lookup(buf);
      if (retval){
            fprintf(stderr,"Failed      to      find      application
%s\n",app.name);
            error();
      }
      fprintf(stderr,"Appid      get_id():      %d,      app.id      :
%d\n",app.get_id(),app.id);
}


void boinc_pw_load(char *login, char *ciphertext, char *gecos, char
*home,
            char *UID, char *GID, char *shell, char *format){
      fprintf(stderr,"Entering boinc_pw_load\n");
      DB_PASSWD pw;
      pw.clear();
      pw.load_pw(login,ciphertext,gecos,home,UID,GID,shell,format);
      fprintf(stderr,"Leaving boinc_pw_load\n");

}//end boinc_load_pw

int pw_check_dup(char *login,char *password_hash,char *format){
      DB_PASSWD pw;
      pw.clear();
      return pw.check_dup(login,password_hash,format);
}//end pw_check_dup

/*Load all passwords into linked Listed PASSWD_ITEMs
 based   on   a   search   Criteria   of   Password   NOT_WORKED   and
BATCH_NOT_SET*/
int  enumerate(std::vector<PASSWD_ITEM>  &target,   int  btch_id,  int
stat_flg){
    DB_PASSWD_ITEM_SET new_set;
    return new_set.enumerate(target,btch_id,stat_flg);
}
```

```
//The following two functions are for trouble shooting
void             reload(std::vector<PASSWD_ITEM>             &target,
std::vector<PASSWD_ITEM> &source){
    while(!source.empty()){
        PASSWD_ITEM new_item;
        new_item.clear();
        new_item=source.back();
        source.pop_back();
        target.push_back(new_item);

    }

}
void print_err(std::vector<PASSWD_ITEM> &source){

    std::vector<PASSWD_ITEM> target;

    while (!source.empty()){
        PASSWD_ITEM new_item;
        new_item.clear();
        new_item=source.back();
        source.pop_back();
/*         fprintf(stderr,"pw.batch_id=%d"
               " ,new_item.pw.password=%s, new_item.pw.format= %s\n"
               ,new_item.pw.batch_id, new_item.pw.password
               ,new_item.pw.format);*/
         target.push_back(new_item);
    }
    reload(source,target);

}

void batch_assign(){
    DB_BATCH batch;
    DB_PASSWD_ITEM_SET pw_stat;
    std::vector<PASSWD_ITEM> not_set_pwd;
    not_set_pwd.clear();
    int found_batch = 0;
 //   batch.clear();

    if (enumerate(not_set_pwd,BATCH_NOT_SET, NOT_WORKED)){
        fprintf(stderr,"Failed    to    load    new    passwords    to
PASSWD_ITEM_SET\n");
        return;
    }
    print_err(not_set_pwd);
    found_batch =batch.check_NOT_WORK(not_set_pwd[0].pw.format);
    if ( found_batch == -1)
    {
//        print_err(not_set_pwd);
        batch.create_batch(DEFAULT_PRIORITY);
        while(!not_set_pwd.empty()){
            PASSWD_ITEM new_item;
            new_item.clear();
```

```
            new_item=not_set_pwd.back();
            not_set_pwd.pop_back();
            new_item.pw.batch_id=batch.id;
            if(pw_stat.update_passwd(new_item.pw))
                fprintf(stderr,"failed to update passwords\n");
        }

    }else{
         while(!not_set_pwd.empty()){
            PASSWD_ITEM new_item;
            new_item.clear();
            new_item=not_set_pwd.back();
            not_set_pwd.pop_back();
            new_item.pw.batch_id=found_batch;
            if(pw_stat.update_passwd(new_item.pw))
                fprintf(stderr,"failed to update passwords\n");
        }
    }

}
```

D. JOHN_CONFIG.H

```
/*
 * This file is part of John the Ripper password cracker,
 * Copyright (c) 1996-2000 by Solar Designer
 */

/*
 * Configuration file loader.
 */

#ifndef _JOHN_CONFIG_H
#define _JOHN_CONFIG_H

/*
 * Parameter list entry.
 */
struct cfg_param {
      struct cfg_param *next;
      char *name, *value;
};

/*
 * Line list entry.
 */
struct cfg_line {
      struct cfg_line *next;
      char *data;
      int number;
};
```

```
/*
 * Main line list structure, head is used to start scanning the
list, while
 * tail is used to add new entries.
 */
struct cfg_list {
      struct cfg_line *head, *tail;
};

/*
 * Section list entry.
 */
struct cfg_section {
      struct cfg_section *next;
      char *name;
      struct cfg_param *params;
      struct cfg_list *list;
};

/*
 * Name of the currently loaded configuration file, or NULL for
none.
 */
extern char *cfg_name;

/*
 * Loads a configuration file, or does nothing if one is already
loaded.
 */
extern void cfg_init(char *name, int allow_missing);

/*
 * Searches for a section with the supplied name, and returns its
line list
 * structure, or NULL if the search fails.
 */
extern   struct   cfg_list   *cfg_get_list(char   *section,   char
*subsection);

/*
 * Searches for a section with the supplied name and a parameter
within the
 * section, and returns the parameter's value, or NULL if not found.
 */
extern char *cfg_get_param(char *section, char *subsection, char
*param);

/*
 * Similar to the above, but does an atoi(). Returns -1 if not
found.
 */
extern  int  cfg_get_int(char  *section,  char  *subsection,  char
*param);
```

```
/*
 * Converts the value to boolean. Returns 0 (false) if not found.
 */
extern int cfg_get_bool(char *section, char *subsection, char
*param);

#endif
```

E. JOHN_CONFIG.C

```
/*
 * This file is part of John the Ripper password cracker,
 * Copyright (c) 1996-2002 by Solar Designer
 */

#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <ctype.h>
#include <errno.h>

#include "misc.h"
#include "params.h"
#include "path.h"
#include "memory.h"
#include "john_config.h"

char *cfg_name = NULL;
static struct cfg_section *cfg_database = NULL;

static char *trim(char *s)
{
      char *e;

      while (*s && (*s == ' ' || *s == '\t')) s++;
      if (!*s) return s;

      e = s + strlen(s) - 1;
      while (e >= s && (*e == ' ' || *e == '\t')) e--;
      *++e = 0;
      return s;
}

static void cfg_add_section(char *name)
{
      struct cfg_section *last;

      last = cfg_database;
      cfg_database = mem_alloc_tiny(
            sizeof(struct cfg_section), MEM_ALIGN_WORD);
      cfg_database->next = last;

      cfg_database->name = str_alloc_copy(name);
```

```
	cfg_database->params = NULL;

	if (!strncmp(name, "list.", 5)) {
		cfg_database->list = mem_alloc_tiny(
			sizeof(struct cfg_list), MEM_ALIGN_WORD);
		cfg_database->list->head  =  cfg_database->list->tail  =
NULL;
	} else {
		cfg_database->list = NULL;
	}
}

static void cfg_add_line(char *line, int number)
{
	struct cfg_list *list;
	struct cfg_line *entry;

	entry	=	mem_alloc_tiny(sizeof(struct	cfg_line),
MEM_ALIGN_WORD);
	entry->next = NULL;

	entry->data = str_alloc_copy(line);
	entry->number = number;

	list = cfg_database->list;
	if (list->tail)
		list->tail = list->tail->next = entry;
	else
		list->tail = list->head = entry;
}

static void cfg_add_param(char *name, char *value)
{
	struct cfg_param *current, *last;

	last = cfg_database->params;
	current = cfg_database->params = mem_alloc_tiny(
		sizeof(struct cfg_param), MEM_ALIGN_WORD);
	current->next = last;

	current->name = str_alloc_copy(name);
	current->value = str_alloc_copy(value);
}

static int cfg_process_line(char *line, int number)
{
	char *p;

	line = trim(line);
	if (!*line || *line == '#' || *line == ';') return 0;

	if (*line == '[') {
		if ((p = strchr(line, ']'))) *p = 0; else return 1;
		cfg_add_section(strlwr(trim(line + 1)));
```

```
	} else
	if (cfg_database && cfg_database->list) {
		cfg_add_line(line, number);
	} else
	if (cfg_database && (p = strchr(line, '='))) {
		*p++ = 0;
		cfg_add_param(strlwr(trim(line)), trim(p));
	} else {
		return 1;
	}

	return 0;
}

static void cfg_error(char *name, int number)
{
	fprintf(stderr, "Error in %s at line %d\n",
		path_expand(name), number);
	error();
}

void cfg_init(char *name, int allow_missing)
{
	FILE *file;
	char line[LINE_BUFFER_SIZE];
	int number;

	if (cfg_database) return;

	if (!(file = fopen(path_expand(name), "r"))) {
		if (allow_missing && errno == ENOENT) return;
		pexit("fopen: %s", path_expand(name));
	}

	number = 0;
	while (fgetl(line, sizeof(line), file))
	if (cfg_process_line(line, ++number)) cfg_error(name, number);

	if (ferror(file)) pexit("fgets");

	if (fclose(file)) pexit("fclose");

	cfg_name = str_alloc_copy(path_expand(name));
}

static struct cfg_section *cfg_get_section(char *section, char
*subsection)
{
	struct cfg_section *current;
	char *p1, *p2;

	if ((current = cfg_database))
	do {
		p1 = current->name; p2 = section;
```

```
            while (*p1 && *p1 == tolower(*p2)) {
                  p1++; p2++;
            }
            if (*p2) continue;

            if ((p2 = subsection))
            while (*p1 && *p1 == tolower(*p2)) {
                  p1++; p2++;
            }
            if (*p1) continue;
            if (p2) if (*p2) continue;

            return current;
      } while ((current = current->next));

      return NULL;
}

struct cfg_list *cfg_get_list(char *section, char *subsection)
{
      struct cfg_section *current;

      if ((current = cfg_get_section(section, subsection)))
            return current->list;

      return NULL;
}

char *cfg_get_param(char *section, char *subsection, char *param)
{
      struct cfg_section *current_section;
      struct cfg_param *current_param;
      char *p1, *p2;

      if ((current_section = cfg_get_section(section, subsection)))
      if ((current_param = current_section->params))
      do {
            p1 = current_param->name; p2 = param;
            while (*p1 && *p1 == tolower(*p2)) {
                  p1++; p2++;
            }
            if (*p1 || *p2) continue;

            return current_param->value;
      } while ((current_param = current_param->next));

      return NULL;
}

int cfg_get_int(char *section, char *subsection, char *param)
{
      char *s_value, *error;
      long l_value;
```

```
      if ((s_value = cfg_get_param(section, subsection, param))) {
            l_value = strtol(s_value, &error, 10);
            if (!*s_value || *error || (l_value & ~0x3FFFFFFFL))
                  return -1;
            return (int)l_value;
      }

      return -1;
}

int cfg_get_bool(char *section, char *subsection, char *param)
{
      char *value;

      if ((value = cfg_get_param(section, subsection, param)))
      switch (*value) {
      case 'y':
      case 'Y':
      case 't':
      case 'T':
      case '1':
            return 1;
      }

      return 0;
}
```

F. JOHN_DB.H

```
/*this file is the header file for boinc.c it contains
 *all structures, classes,  and functions dealing
 *with database interactions
 */
#ifndef _JOHN_DB_H
#define _JOHN_DB_H

//#include <cstdio>
//#include <vector>

#include <mysql.h>
#include <db_base.h>
#include <boinc_db.h>
#include "boinc.h"

extern DB_CONN john_db;


struct PASSWD {
    int id;
    char *login;
    char *passwd_hash;
    int UID;
    int GID;
```

```
    char *GECOS;
    char *home_dir;
    char *shell;
    char *password;
    char *format;
    int batch_id;

    void clear();
};

struct STATUS{
    int id;
    int pwid;
    int status_flag;
    int priority;
    unsigned long workunits_gen;
    unsigned long workunits_done;

    void clear();
};
/*
struct BATCH{
    int id;
    int mode_flag;
    int batch_priority;
    int s_loc;//s=start ... Entry or Rule #
    int s_length;
    int s_fixed;
    int s_count;
    int s_num0;
    int s_num1;
    int s_num2;
    int s_num3;
    int s_num4;
    int s_num5;
    int s_num6;
    int s_num7;
    int f_loc;
    int f_length;
    int f_fixed;
    int f_count;
    int f_num0;
    int f_num1;
    int f_num2;
    int f_num3;
    int f_num4;
    int f_num5;
    int f_num6;
    int f_num7;

    void clear();
};*/
struct CHECK_ITEM{
      int id;
```

```
      int t_nresult;

      void clear();
      void parse(MYSQL_ROW&);
};

struct PASSWD_ITEM {
      PASSWD pw;
      STATUS pw_stat;

      void clear();
      void parse(MYSQL_ROW&);
    PASSWD_ITEM& operator=(const PASSWD_ITEM &b);
};

void batch_equal(BATCH &t, BATCH &s);


class DB_PASSWD : public DB_BASE, public PASSWD{
public:
    DB_PASSWD(DB_CONN* p=0);
    int get_id();
    void db_print(char*);
    void db_parse(MYSQL_ROW &row);
    int check_dup(char*, char*,char*);
    void load_pw(char*,char*,char*,char*,char*,char*,char*,char*);

};

class DB_STATUS : public DB_BASE, public STATUS{
public:
    DB_STATUS(DB_CONN* p=0);
    int get_id();
    void db_print(char*);
    void db_parse(MYSQL_ROW &row);
    int check_flag(int flg);
    void operator=(STATUS& r) {STATUS::operator=(r);}
};

class DB_BATCH : public DB_BASE, public BATCH{
public:
    DB_BATCH(DB_CONN* p=0);
    int get_id();
    void db_print(char*);
    void db_parse(MYSQL_ROW &row);
    int check_for_batch(int,int,int,char*);
    void create_batch(int);
    int update_ruleid(int);
    int check_NOT_WORK(char*);
};

class DB_PASSWD_ITEM_SET: public DB_BASE_SPECIAL{
public:
      DB_PASSWD_ITEM_SET(DB_CONN* p=0);
```

```
      PASSWD_ITEM last_item;
      int items_this_q;

      int  enumerate(std::vector<PASSWD_ITEM>&  items,  int  btch_id,
int stat_flg );
//    int  enumerate_test(std::vector<PASSWD_ITEM>& items,  char  *op,
int btch_id, int stat_flg );
      int update_status(STATUS&);
//    int  update_passwds(std::vector<PASSWD_ITEM>& items,  int  flag,
int ruleid);
      int update_passwd(PASSWD &pw);
      int get_passwd(int,char*);
      int update_pw(char*);
};



#endif
```

G. JOHN_DB.C

```
#include <cstdlib>
#include <cstring>
#include <ctime>
#include <unistd.h>
#include <stdio.h>
#include <sched_config.h>
#include <util.h>
#include <error_numbers.h>
#include "john_db.h"
#include "boinc.h"

extern "C" {
//#include "logger.h"
#include "memory.h"
//#include "misc.h"
#include "params.h"
}
using namespace std;

//SCHED_CONFIG config;
DB_CONN john_db;

//static struct PWLIST *passwd_list = NULL;

void PASSWD::clear() {memset(this, 0, sizeof(*this));}
void STATUS::clear() {memset(this, 0, sizeof(*this));}
//void BATCH::clear() {memset(this, 0, sizeof(*this));}
void CHECK_ITEM::clear() {memset(this, 0, sizeof(*this));}
void PASSWD_ITEM::clear() {
      memset(this, 0, sizeof(*this));
      pw.clear();
      pw_stat.clear();
```

```
}

DB_PASSWD::DB_PASSWD(DB_CONN* dc) :
      //DB_PASSWD("passwords", dc?dc:&john_db){}
      DB_BASE((const char*) "passwords", dc?dc:&john_db){}
DB_STATUS::DB_STATUS(DB_CONN* dc) :
      //DB_STATUS("status", dc?dc:&john_db){}
      DB_BASE("status", dc?dc:&john_db){}
DB_BATCH::DB_BATCH(DB_CONN* dc) :
      //DB_BATCH("batch", dc?dc:&john_db){}
      DB_BASE("batch", dc?dc:&john_db){}
DB_PASSWD_ITEM_SET::DB_PASSWD_ITEM_SET(DB_CONN* dc):
      //DB_PASSWD_ITEM_SET(dc?dc:&john_db){}
      DB_BASE_SPECIAL(dc?dc:&john_db){}


int DB_PASSWD::get_id() {return id;}
int DB_STATUS::get_id() {return id;}
int DB_BATCH::get_id() {return id;}


void DB_PASSWD::db_print(char* buf){
   sprintf(buf,"id=%d, login='%s', passwd_hash='%s',"
      "userid='%d',      groupid='%d',      GECOS='%s',home_dir='%s',
shell='%s',"
      "password='%s',format='%s',batch_id='%d'",      id,      login,
passwd_hash, UID, GID,
      GECOS, home_dir, shell, password, format,batch_id);
}



int DB_PASSWD::check_dup(char *l, char *pw_hash,char *f){
      char query[MAX_QUERY_LEN];
      int retval;
      MYSQL_RES *res;
      MYSQL_ROW row;

      sprintf(query,"Select * from passwords where login='%s'"
                  "   and   passwd_hash='%s'   and   format='%s'",   l,
pw_hash,f);
      retval=db->do_query(query);
      if (retval) return mysql_errno(db->mysql);
      res = mysql_store_result(db->mysql);
      if (!res) return mysql_errno(db->mysql);
      row = mysql_fetch_row(res);
      if (row)
            return 0;
      return 1;
}
int DB_BATCH::check_NOT_WORK(char *form){
    char query[MAX_QUERY_LEN];
    int retval;
    MYSQL_RES *res;
```

```
    MYSQL_ROW row;
    PASSWD_ITEM pi;
    pi.clear();
//    fprintf(stderr,"IN Check_NOT_WORKED\n");
    sprintf(query, "Select * from passwords AS pw"
                  " LEFT JOIN status AS stat ON pw.id = stat.pwid"
                  "     WHERE     pw.batch_id     !=     '%d'     and
stat.status_flag='%d'"
                 " and pw.format='%s' LIMIT 1 "
                 ,BATCH_NOT_SET,NOT_WORKED,form);
    retval=db->do_query(query);
//    fprintf(stderr,"retval=%d\n",retval);
    if (retval) return mysql_errno(db->mysql);
    res= mysql_store_result(db->mysql);
    if (!res) return mysql_errno(db->mysql);
    row = mysql_fetch_row(res);
    if (row){
        pi.parse(row);
//                                    fprintf(stderr,"b_ID    %d:form
%s\n",pi.pw.batch_id,pi.pw.format);
        return pi.pw.batch_id;
    }
    return -1;
}
void DB_PASSWD::db_parse(MYSQL_ROW &r){
      int i=0;
      clear();
      id=atol(r[i++]);
      login=r[i++];
      passwd_hash=r[i++];
      UID=atol(r[i++]);
      GID=atol(r[i++]);
      GECOS=r[i++];
      home_dir=r[i++];
      shell=r[i++];
      password=r[i++];
      format=r[i++];
    batch_id=atoi(r[i++]);
}


void CHECK_ITEM::parse(MYSQL_ROW &r){
      fprintf(stderr,"inside parse\n");
      int i=0;
      clear();
      fprintf(stderr,"after clear: \n");
      id=atoi(r[i++]);
      fprintf(stderr,"after id assignment\n");
      t_nresult=atoi(r[i++]);
}

void DB_STATUS::db_print(char* buf){
   sprintf(buf,
        "id='%d', pwid='%d', status_flag='%d', priority='%d',"
```

```
        "workunits_gen='%d', workunits_done='%d'", id, pwid,
      status_flag, priority, workunits_gen, workunits_done);
}

void DB_STATUS::db_parse(MYSQL_ROW &r){
      int i=0;
      clear();
      id=atol(r[i++]);
      pwid=atol(r[i++]);
      status_flag=atol(r[i++]);
      priority=atol(r[i++]);
      workunits_gen=atol(r[i++]);
      workunits_done=atol(r[i++]);
}
/*this function returns true if an object
in the status table has a flag=flg*/
int DB_STATUS::check_flag(int flg){
      char query[MAX_QUERY_LEN];
      int retval;
      MYSQL_ROW row;
      MYSQL_RES *res;
      sprintf(query,"Select * from status where status_flag = '%d'",
flg );
      retval = db->do_query(query);
      if (retval) return mysql_errno(db->mysql);

      res = mysql_store_result(db->mysql);
      if (!res) return mysql_errno(db->mysql);

      row = mysql_fetch_row(res);
      if (!row){
            mysql_free_result(res);
            return 0;
      }
      return 1;
}

void DB_BATCH::db_print(char *buf){
      sprintf(buf," id='%d', mode_flag='%d', batch_priority='%d',"
        "s_loc='%d', s_length='%d', s_fixed='%d', s_count='%d',"
        "s_num0='%d', s_num1='%d', s_num2='%d', s_num3='%d',"
        "s_num4='%d', s_num5='%d', s_num6='%d', s_num7='%d',"
        "f_loc='%d', f_length='%d', f_fixed='%d', f_count='%d',"
        "f_num0='%d', f_num1='%d', f_num2='%d', f_num3='%d',"
        "f_num4='%d', f_num5='%d', f_num6='%d', f_num7='%d'"
        ,id,mode_flag,batch_priority,s_loc,s_length,s_fixed,s_count
        ,s_num0,s_num1,s_num2,s_num3,s_num4,s_num5,s_num6,s_num7

,f_loc,f_length,f_fixed,f_count,f_num0,f_num1,f_num2,f_num3,f_num4
        ,f_num5,f_num6,f_num7);

}
void batch_equal(BATCH &t, BATCH &s){
   t.id=s.id;
```

```
   t.mode_flag=s.mode_flag;
   t.batch_priority=s.batch_priority;
   t.s_loc=s.s_loc;//s=start ... Entry or Rule #
   t.s_length=s.s_length;
   t.s_fixed=s.s_fixed;
   t.s_count=s.s_count;
   t.s_num0=s.s_num0;
   t.s_num1=s.s_num1;
   t.s_num2=s.s_num2;
   t.s_num3=s.s_num3;
   t.s_num4=s.s_num4;
   t.s_num5=s.s_num5;
   t.s_num6=s.s_num6;
   t.s_num7=s.s_num7;
   t.f_loc=s.f_loc;
   t.f_length=s.f_length;
   t.f_fixed=s.f_fixed;
   t.f_count=s.f_count;
   t.f_num0=s.f_num0;
   t.f_num1=s.f_num1;
   t.f_num2=s.f_num2;
   t.f_num3=s.f_num3;
   t.f_num4=s.f_num4;
   t.f_num5=s.f_num5;
   t.f_num6=s.f_num6;
   t.f_num7=s.f_num7;

}
void DB_BATCH::db_parse(MYSQL_ROW &r){
    int i=0;
 //   b_clear(*this);
    id=atoi(r[i++]);
    mode_flag=atoi(r[i++]);
    batch_priority =atoi(r[i++]);
    s_loc=atoi(r[i++]);
    s_length=atoi(r[i++]);
    s_fixed=atoi(r[i++]);
    s_count=atoi(r[i++]);
    s_num0=atoi(r[i++]);
    s_num1=atoi(r[i++]);
    s_num2=atoi(r[i++]);
    s_num3=atoi(r[i++]);
    s_num4=atoi(r[i++]);
    s_num5=atoi(r[i++]);
    s_num6=atoi(r[i++]);
    s_num7=atoi(r[i++]);
    f_loc=atoi(r[i++]);
    f_length=atoi(r[i++]);
    f_fixed=atoi(r[i++]);
    f_count=atoi(r[i++]);
    f_num0=atoi(r[i++]);
    f_num1=atoi(r[i++]);
    f_num2=atoi(r[i++]);
    f_num3=atoi(r[i++]);
```

```
    f_num4=atoi(r[i++]);
    f_num5=atoi(r[i++]);
    f_num6=atoi(r[i++]);
    f_num7=atoi(r[i++]);
}

void DB_BATCH::create_batch(int p){
    fprintf(stderr,"New batch created\n");
    id=0;
    mode_flag=SINGLE_MODE;
    batch_priority=p;
    s_loc=0;
    s_length=0;
    s_fixed=0;
    s_count=0;
    s_num0=0;
    s_num1=0;
    s_num2=0;
    s_num3=0;
    s_num4=0;
    s_num5=0;
    s_num6=0;
    s_num7=0;
    f_loc=0;
    f_length=0;
    f_fixed=0;
    f_count=0;
    f_num0=0;
    f_num1=0;
    f_num2=0;
    f_num3=0;
    f_num4=0;
    f_num5=0;
    f_num6=0;
    f_num7=0;
    if(insert()){
           fprintf(stderr,"failed to insert Batch\n");
    }
    id=db->insert_id();
}
    /*this is for the creation of unique input file names*/
int DB_BATCH::check_for_batch(int f,int p,int ruleid,char *form){
    char buf[256];
    int retval;
    MYSQL_ROW row;
    MYSQL_RES *res;

    sprintf(buf,"select * from %s where flag='%d' and priority='%d'"
           "and batch_ruleid='%d' and format='%s'",
           table_name, f,p,ruleid,form);
    fprintf(stderr,"%s\n",buf);
    retval = db->do_query(buf);
    if (retval) return mysql_errno(db->mysql);
    res= mysql_store_result(db->mysql);
```

```
    if (!res) return mysql_errno(db->mysql);
    row = mysql_fetch_row(res);
    if (!row){
      return 1;
    }
     db_parse(row);
     return 0;
}

int DB_BATCH::update_ruleid(int ruleid){
      char buf[256];
      sprintf(buf,"batch_ruleid=%d",ruleid);
        return update_field(buf);
}


void PASSWD_ITEM::parse(MYSQL_ROW &r){
/*fill in the fields I want to parse from
the rows returned in the enumerate function*/
    int i=0;
    clear();
    pw.id=atoi(r[i++]);
    pw.login=r[i++];
    pw.passwd_hash=r[i++];
    pw.UID=atoi(r[i++]);
    pw.GID=atoi(r[i++]);
    pw.GECOS=r[i++];
    pw.home_dir=r[i++];
    pw.shell=r[i++];
    pw.password=r[i++];
    pw.format=r[i++];
    pw.batch_id=atoi(r[i++]);
    pw_stat.id=atoi(r[i++]);
    pw_stat.pwid=atoi(r[i++]);
    pw_stat.status_flag=atoi(r[i++]);
    pw_stat.priority=atoi(r[i++]);
    pw_stat.workunits_gen=atoi(r[i++]);
    pw_stat.workunits_done=atoi(r[i++]);
}
PASSWD_ITEM&  PASSWD_ITEM::operator=(const PASSWD_ITEM &b){
      pw.id=b.pw.id;
      pw.login = new char[254];
      strcpy(pw.login,b.pw.login);
      pw.passwd_hash = new char[254];
      strcpy(pw.passwd_hash,b.pw.passwd_hash);
      pw.UID=b.pw.UID;
      pw.GID=b.pw.GID;
      pw.GECOS = new char[254];
      strcpy(pw.GECOS,b.pw.GECOS);
      pw.home_dir = new char[254];
      strcpy(pw.home_dir, b.pw.home_dir);
      pw.shell = new char[254];
      strcpy(pw.shell,b.pw.shell);
      pw.password = new char[254];
```

```
      strcpy(pw.password,b.pw.password);
      pw.format = new char[254];
      strcpy(pw.format,b.pw.format);
    pw.batch_id=b.pw.batch_id;
      pw_stat.id=b.pw_stat.id;
      pw_stat.pwid=b.pw_stat.pwid;
      pw_stat.status_flag=b.pw_stat.status_flag;
      pw_stat.priority=b.pw_stat.priority;
      pw_stat.workunits_gen=b.pw_stat.workunits_gen;
      pw_stat.workunits_done=b.pw_stat.workunits_done;
      return *this;

}
void  DB_PASSWD::load_pw(char*  l,  char  *pw_hash,  char  *gecos,char
*home,
                 char *uid, char *gid, char *SHELL, char *form){
      int retval;
      DB_STATUS stat;

      id=0;
        fprintf(stderr,"id:%d\n",id);
        fprintf(stderr,"passed in var:%s\n",l);
//      strcpy2(login,login);
        login=l;
        fprintf(stderr,"login:%s\n",login);
        passwd_hash=pw_hash;
        fprintf(stderr,"password_hash:%s\n",passwd_hash);
        UID=atoi(uid);
        GID=atoi(gid);
        GECOS=gecos;
        home_dir=home;
        shell=SHELL;
          password="NO_PASSWORD";
          format=form;
        batch_id=BATCH_NOT_SET;
        retval=insert();
        fprintf(stderr,"%d:%s:%s:%d:%d:%s:%s:%s:%s\n", id,login,
                        passwd_hash,UID,GID,GECOS,
                        home_dir,shell,format);
        if(retval){
                //log_event("Failed     to     insert     pw     into
%s",table_name);
                fprintf(stderr,"Failed    to    insert    pw    into
%s",table_name);
                //error();
        }
        //this retrieves the last thing inserted to the DB
        id=db->insert_id();
        stat.clear();
        stat.id=0;
        stat.pwid=id;
        stat.status_flag=NOT_WORKED;
        stat.priority=DEFAULT_PRIORITY;
        stat.workunits_gen=0;
```

```
        stat.workunits_done=0;
        retval=stat.insert();
        if (retval){
               //    log_event("Failed    to    insert    stat    into
%s",stat.table_name);
                fprintf(stderr,"Failed    to    insert    stat    into
%s",stat.table_name);
                //error();
      }


}
// useful, but didn't use
int DB_PASSWD_ITEM_SET::get_passwd(int pw_id,char *pw_hash){
      char query[MAX_QUERY_LEN];
      int retval;
      MYSQL_RES *res;
      MYSQL_ROW row;
      fprintf(stderr,"pw_hash: %s\n",pw_hash);
      sprintf(query,"Select * from passwords as pw "
                "left join status as st on pw.id=st.pwid where"
                "    pw.id    =    '%d'    and    pw.passwd_hash    =
'%s'",pw_id,pw_hash);
      retval = db->do_query(query);
      if (retval) return mysql_errno(db->mysql);
      res = mysql_store_result(db->mysql);
      if (!res) return mysql_errno(db->mysql);
      row = mysql_fetch_row(res);
      last_item.parse(row);
      return 0;

}
// didn't use ... look at more before delete
int DB_PASSWD_ITEM_SET::update_pw(char *plain){
      char *pass;
      pass = str_alloc_copy(plain);
      fprintf(stderr,"pass: %s plaine: %s\n",pass,plain);
      if (!(strncmp(pass,"NO_PASSWORD",11))){
            return 0;
      }else{
            last_item.pw.password = pass;
            last_item.pw_stat.status_flag = CRACKED;
            if (update_passwd(last_item.pw)) return 1;
            if (update_status(last_item.pw_stat)) return 1;
      }
      return 0;
}



/*This function Queries the John DB for all passwords and there
status  by  using  a  Left  outer  Join  on  the  tables  passwords  and
status.
```

```
Then the attributes asked for are stored into a VECTOR of
PASSWD_ITEMs
This vector will be used by the work generator to create workunits
for
a particular password hash*/
int          DB_PASSWD_ITEM_SET::enumerate(std::vector<PASSWD_ITEM>&
items,int btch_id, int stat_flg){
      int retval;
      char query[MAX_QUERY_LEN];
      MYSQL_ROW row;

      fprintf(stderr,"In enum\n");
      if (!cursor.active){
      sprintf(query, "Select * from passwords AS pw "
                  "  LEFT JOIN status AS stat ON pw.id = stat.pwid "
                  "         WHERE    pw.batch_id    =    '%d'    and
stat.status_flag='%d' "
                ,btch_id,stat_flg);

//          fprintf(stderr,"%s\n",query);
        retval = db->do_query(query);

            if (retval) return mysql_errno(db->mysql);

            cursor.rp = mysql_store_result(db->mysql);
            if (!cursor.rp) return mysql_errno(db->mysql);
            cursor.active = true;
            if(cursor.active)
                  fprintf(stderr,"cursor active\n");

            row = mysql_fetch_row(cursor.rp);
            if (!row){
                  mysql_free_result(cursor.rp);
                  cursor.active = false;
                  return 1;
            }
//        last_item.clear();
//          last_item.parse(row);
            items_this_q = 0;
      }
    last_item.clear();
      items.clear();
      while (true) {
        PASSWD_ITEM new_item;
        new_item.clear();
        new_item.parse(row);
        items.push_back(new_item);
        last_item=new_item;
        items_this_q++;
            fprintf(stderr,"%s:%s:%s:%d:%d:%d\n",last_item.pw.login,
                        last_item.pw.passwd_hash,
                  last_item.pw.password,
                        last_item.pw_stat.status_flag,
                        last_item.pw_stat.priority,
```

```
                        last_item.pw.batch_id);
            row = mysql_fetch_row(cursor.rp);
            if (!row) {
                  mysql_free_result(cursor.rp);
                  cursor.active = false;
                  return 0;
            }
    }

      return 0;
}
/*int   DB_PASSWD_ITEM_SET::enumerate_test(std::vector<PASSWD_ITEM>&
items,char op,int btch_id, int stat_flg){
      int retval;
      char query[MAX_QUERY_LEN];
      MYSQL_ROW row;
      last_item.clear();
      items.clear();

      fprintf(stderr,"In enum\n");
      if (!cursor.active){
            sprintf(query, "Select * from passwords AS pw "
                  "  LEFT JOIN status AS stat ON pw.id = stat.pwid "
                  "        WHERE    pw.batch_id    %s=    '%d'    and
stat.status_flag=%d "
             ,op,btch_id,stat_flg);
            retval = db->do_query(query);

            if (retval) return mysql_errno(db->mysql);

            cursor.rp = mysql_store_result(db->mysql);
            if (!cursor.rp) return mysql_errno(db->mysql);
            cursor.active = true;
            if(cursor.active)
                  fprintf(stderr,"cursor active\n");

            row = mysql_fetch_row(cursor.rp);
            if (!row){
                  mysql_free_result(cursor.rp);
                  cursor.active = false;
                  return 1;
            }
//        last_item.clear();
//         last_item.parse(row);
            items_this_q = 0;
      }
    while (true) {
        PASSWD_ITEM new_item;
        new_item.clear();
        new_item.parse(row);
        items.push_back(new_item);
        last_item=new_item;
        items_this_q++;
            fprintf(stderr,"%s:%s:%s:%d:%d:%d\n",last_item.pw.login,
```

```
                        last_item.pw.passwd_hash,
                  last_item.pw.password,
                        last_item.pw_stat.status_flag,
                        last_item.pw_stat.priority,
                        last_item.pw.batch_id);
            row = mysql_fetch_row(cursor.rp);
            if (!row) {
                  mysql_free_result(cursor.rp);
                  cursor.active = false;
                  return 0;
            }
    }

      return 0;
}
*/
//for updating a STATUS object
//used by workgenerator
int DB_PASSWD_ITEM_SET::update_status(STATUS &stat){
      char query[MAX_QUERY_LEN];
//    fprintf(stderr,"Inside update_status\n");
      sprintf(query,"update     status     set     status_flag='%d',
priority='%d',"
            "workunits_gen='%d', workunits_done='%d' where id='%d'"
        , stat.status_flag, stat.priority, stat.workunits_gen
        , stat.workunits_done,stat.id);
      return db->do_query(query);
}

//for updating passwords when cracked
//used by validator...
int DB_PASSWD_ITEM_SET::update_passwd(PASSWD &pw){
        char query[MAX_QUERY_LEN];
    fprintf(stderr,"password=%s, batch_id=%d, pwid=%d\n",pw.password
            , pw.batch_id, pw.id);
      sprintf(query,"update     passwords     set     password='%s',
batch_id='%d' where "
            "id='%d'", pw.password, pw.batch_id, pw.id);
      fprintf(stderr,"%s\n",query);
      return db->do_query(query);
}
/*//Probably going to get rid of this...
int     DB_PASSWD_ITEM_SET::update_passwds(std::vector<PASSWD_ITEM>&
items,int flag, int ruleid){
      char query[MAX_QUERY_LEN];
//    PASSWD_ITEM *cur;

      for (int i=0;i<items.size();i++){
            items[i].pw_stat.status_flag=flag;
            items[i].pw_stat.last_ruleid=ruleid;
            if (update_status(items[i].pw_stat))
                  return 1;
      }
      return 0;
```

```
}*/
```

H. OPTIONS.H

```
/*
 * This file is part of John the Ripper password cracker,
 * Copyright (c) 1996-98,2003 by Solar Designer
 */

/*
 * John's command line options definition.
 */

#ifndef _JOHN_OPTIONS_H
#define _JOHN_OPTIONS_H

#include "list.h"
#include "loader.h"
#include "getopt.h"

/*
 * Option flags bitmasks.
 */
/* An action requested */
#define FLG_ACTION                0x00000001
/* Password files specified */
#define FLG_PASSWD                0x00000002
/* An option supports password files */
#define FLG_PWD_SUP               0x00000004
/* An option requires password files */
#define FLG_PWD_REQ               (0x00000008 | FLG_PWD_SUP)
/* Some option that doesn't have its own flag is specified */
#define FLG_NONE            0x00000010
/* A cracking mode enabled */
#define FLG_CRACKING_CHK          0x00000020
#define FLG_CRACKING_SUP          0x00000040
#define FLG_CRACKING_SET \
      (FLG_CRACKING_CHK   |   FLG_CRACKING_SUP   |   FLG_ACTION   |
FLG_PWD_REQ)
/* Wordlist mode enabled, options.wordlist is set to the file name
or NULL
 * if reading from stdin. */
#define FLG_WORDLIST_CHK          0x00000080
#define FLG_WORDLIST_SET          (FLG_WORDLIST_CHK                |
FLG_CRACKING_SET)
/* Wordlist mode enabled, reading from stdin */
#define FLG_STDIN_CHK             0x00000100
#define FLG_STDIN_SET             (FLG_STDIN_CHK | FLG_WORDLIST_SET)
/* Wordlist rules enabled */
#define FLG_RULES           0x00000200
/* "Single crack" mode enabled */
#define FLG_SINGLE_CHK            0x00000400
```

```
#define FLG_SINGLE_SET            (FLG_SINGLE_CHK                   |
FLG_CRACKING_SET)
/* Incremental mode enabled */
#define FLG_INC_CHK               0x00000800
#define FLG_INC_SET               (FLG_INC_CHK | FLG_CRACKING_SET)
/* External mode or word filter enabled */
#define FLG_EXTERNAL_CHK          0x00001000
#define FLG_EXTERNAL_SET \
      (FLG_EXTERNAL_CHK   |   FLG_ACTION   |   FLG_CRACKING_SUP   |
FLG_PWD_SUP)
/* Batch cracker */
#define FLG_BATCH_CHK             0x00004000
#define FLG_BATCH_SET             (FLG_BATCH_CHK | FLG_CRACKING_SET)
/* Stdout mode */
#define FLG_STDOUT                0x00008000
/* Restoring an interrupted session */
#define FLG_RESTORE_CHK                0x00010000
#define FLG_RESTORE_SET                (FLG_RESTORE_CHK           |
FLG_ACTION)
/* A session name is set */
#define FLG_SESSION               0x00020000
/* Print status of a session */
#define FLG_STATUS_CHK            0x00040000
#define FLG_STATUS_SET            (FLG_STATUS_CHK | FLG_ACTION)
/* Make a charset */
#define FLG_MAKECHR_CHK                0x00100000
#define FLG_MAKECHR_SET \
      (FLG_MAKECHR_CHK | FLG_ACTION | FLG_PWD_SUP)
/* Show cracked passwords */
#define FLG_SHOW_CHK              0x00200000
#define FLG_SHOW_SET \
      (FLG_SHOW_CHK | FLG_ACTION | FLG_PWD_REQ)
/* Perform a benchmark */
#define FLG_TEST_CHK              0x00400000
#define FLG_TEST_SET \
      (FLG_TEST_CHK | FLG_CRACKING_SUP | FLG_ACTION)
/* Passwords per salt requested */
#define FLG_SALTS           0x01000000
/* Ciphertext format forced */
#define FLG_FORMAT                0x02000000
/* Memory saving enabled */
#define FLG_SAVEMEM               0x04000000
/* Application Name Set*/
#define FLG_APP                   0x10000000

/*
 * Structure with option flags and all the parameters.
 */
struct options_main {
/* Option flags */
      opt_flags flags;

/* Password files */
      struct list_main *passwd;
```

```
/* Password file loader options */
      struct db_options loader;

/* Session name */
      char *session;

/* Ciphertext format name */
      char *format;

/* Wordlist file name */
      char *wordlist;

/* Charset file name */
      char *charset;

/* External mode or word filter name */
      char *external;
/* Application Name */
      char *app_name;

/* Maximum plaintext length for stdout mode */
      int length;
};

extern struct options_main options;

/*
 * Initializes the options structure.
 */
extern void opt_init(int argc, char **argv);

#endif
```

I. OPTIONS.C

```
/*
 * This file is part of John the Ripper password cracker,
 * Copyright (c) 1996-2005 by Solar Designer
 */

#include <stdio.h>
#include <stdlib.h>
#include <string.h>

#include "arch.h"
#include "misc.h"
#include "params.h"
#include "memory.h"
#include "list.h"
#include "loader.h"
#include "logger.h"
#include "status.h"
```

```
#include "recovery.h"
#include "options.h"


struct options_main options;

static struct opt_entry opt_list[] = {
      {"", FLG_PASSWD, 0, 0, 0, OPT_FMT_ADD_LIST, &options.passwd},
      {"app",FLG_APP,0,0,0,OPT_FMT_STR_ALLOC, &options.app_name},
      {"single", FLG_SINGLE_SET, FLG_CRACKING_CHK},
      {"wordlist", FLG_WORDLIST_SET, FLG_CRACKING_CHK,
            0, OPT_REQ_PARAM, OPT_FMT_STR_ALLOC, &options.wordlist},
      {"rules",      FLG_RULES,      FLG_RULES,      FLG_WORDLIST_CHK,
FLG_STDIN_CHK},
      {"incremental", FLG_INC_SET, FLG_CRACKING_CHK,
            0, 0, OPT_FMT_STR_ALLOC, &options.charset},
/*    {"make-charset", FLG_MAKECHR_SET, FLG_MAKECHR_CHK,
            0, FLG_CRACKING_CHK | FLG_SESSION | OPT_REQ_PARAM,
            OPT_FMT_STR_ALLOC, &options.charset},
      {"show", FLG_SHOW_SET, FLG_SHOW_CHK,
            0, FLG_CRACKING_SUP | FLG_MAKECHR_CHK},
      {"test", FLG_TEST_SET, FLG_TEST_CHK,
            0, ~FLG_TEST_SET & ~FLG_FORMAT & ~FLG_SAVEMEM &
            ~OPT_REQ_PARAM},
      {"users", FLG_NONE, 0, FLG_PASSWD, OPT_REQ_PARAM,
            OPT_FMT_ADD_LIST_MULTI, &options.loader.users},
      {"groups", FLG_NONE, 0, FLG_PASSWD, OPT_REQ_PARAM,
            OPT_FMT_ADD_LIST_MULTI, &options.loader.groups},
      {"shells", FLG_NONE, 0, FLG_PASSWD, OPT_REQ_PARAM,
            OPT_FMT_ADD_LIST_MULTI, &options.loader.shells},
      {"salts", FLG_SALTS, FLG_SALTS, FLG_PASSWD, OPT_REQ_PARAM,
            "%d", &options.loader.min_pps},
      {"format", FLG_FORMAT, FLG_FORMAT,
            FLG_CRACKING_SUP,
            FLG_MAKECHR_CHK | FLG_STDOUT | OPT_REQ_PARAM,
            OPT_FMT_STR_ALLOC, &options.format},
      {"save-memory", FLG_SAVEMEM, FLG_SAVEMEM, 0, OPT_REQ_PARAM,
            "%u", &mem_saving_level},*/
      {NULL}
};

#if DES_BS
/* nonstd.c and parts of x86-mmx.S aren't mine */
#define JOHN_COPYRIGHT \
      "Solar Designer and others"
#else
#define JOHN_COPYRIGHT \
      "Solar Designer"
#endif
//changed
#define JOHN_BOINC_USAGE \
"Distributed John the Ripper password cracker - Work Generator ,
version \
 " JOHN_BOINC_VERSION "\n" \
```

```
"Copyright (c) 1996-2005 by " JOHN_COPYRIGHT "\n" \
"Homepage: \n" \
"\n" \
"Usage: %s [OPTIONS] [PASSWORD-FILES]\n" \
"--single                   \"single crack\" mode\n" \
"--app=NAME          Application Name\n" \
"--wordlist=FILE --stdin     wordlist mode, read words from FILE or
stdin\n" \
"--rules                     enable word mangling rules for wordlist
mode\n" \
"--incremental[=MODE]           \"incremental\" mode [using section
MODE]\n" /*\
"--make-charset=FILE               make  a  charset,  FILE  will  be
overwritten\n" \
"--show                     show cracked passwords\n" \
"--test                     perform a benchmark\n" \
"--users=[-]LOGIN|UID[,..]    [do  not]  load  this  (these)  user(s)
only\n" \
"--groups=[-]GID[,..]            load  users  [not]  of  this  (these)
group(s) only\n" \
"--shells=[-]SHELL[,..]         load  users  with[out]  this  (these)
shell(s) only\n" \
"--salts=[-]COUNT               load salts with[out] at least COUNT
passwords " \
      "only\n" \
"--format=NAME              force ciphertext format NAME: " \
      "DES/BSDI/MD5/BF/AFS/LM\n" \
"--save-memory=LEVEL        enable memory saving, at LEVEL 1..3\n"
*/
void opt_init(int argc, char **argv)
{
      if (!argv[0]) error();

      if (!argv[1]) {
            printf(JOHN_BOINC_USAGE, argv[0]);
            exit(0);
      }

      memset(&options, 0, sizeof(options));

      list_init(&options.passwd);

      options.loader.flags = DB_LOGIN;
      list_init(&options.loader.users);
      list_init(&options.loader.groups);
      list_init(&options.loader.shells);

      options.length = -1;

      opt_process(opt_list, &options.flags, argv);

      if ((options.flags &
          (FLG_EXTERNAL_CHK  |  FLG_CRACKING_CHK  |  FLG_MAKECHR_CHK))
==
```

```
        FLG_EXTERNAL_CHK)
          options.flags |= FLG_CRACKING_SET;

      if (!(options.flags & FLG_ACTION))
            options.flags |= FLG_BATCH_SET;

      opt_check(opt_list, options.flags, argv);

      if (options.session)
            rec_name = options.session;

      if (options.flags & FLG_RESTORE_CHK) {
            rec_restore_args(1);
            return;
      }

      if (options.flags & FLG_STATUS_CHK) {
            rec_restore_args(0);
            options.flags |= FLG_STATUS_SET;
            status_init(NULL, 1);
            status_print();
            exit(0);
      }

      if (options.flags & FLG_SALTS)
      if (options.loader.min_pps < 0) {
            options.loader.max_pps = -1 - options.loader.min_pps;
            options.loader.min_pps = 0;
      }

      if (options.length < 0)
            options.length = PLAINTEXT_BUFFER_SIZE - 3;
      else
      if    (options.length    <    1    ||    options.length    >
PLAINTEXT_BUFFER_SIZE - 3) {
            fprintf(stderr, "Invalid plaintext length requested\n");
            error();
      }

      if (options.flags & FLG_STDOUT) options.flags &= ~FLG_PWD_REQ;

      if   ((options.flags   &   (FLG_PASSWD   |   FLG_PWD_REQ))   ==
FLG_PWD_REQ) {
            fprintf(stderr, "Password files required, "
                  "but none specified\n");
            error();
      }

      if   ((options.flags   &   (FLG_PASSWD   |   FLG_PWD_SUP))   ==
FLG_PASSWD) {
            fprintf(stderr, "Password files specified, "
                  "but no option would use them\n");
            error();
      }
```

```
	rec_argc = argc; rec_argv = argv;
}
```

J. PARAMS.H

```
/*
 * This file is part of John the Ripper password cracker,
 * Copyright (c) 1996-2005 by Solar Designer
 */

/*
 * Some global parameters.
 */

#ifndef _JOHN_PARAMS_H
#define _JOHN_PARAMS_H

#include <limits.h>

/*
 * John's version number.
 */
#define JOHN_VERSION		"1.6.38"
#define JOHN_BOINC_VERSION	"1.0"
/*
 *  Is  this  a  system-wide  installation?  *BSD  ports  and  Linux
distributions
 * will probably want to set this to 1 for their builds of John.
 */
#ifndef JOHN_SYSTEMWIDE
#define JOHN_SYSTEMWIDE		0
#endif

#if JOHN_SYSTEMWIDE
#define JOHN_SYSTEMWIDE_EXEC	"/usr/libexec/john"
#define JOHN_SYSTEMWIDE_HOME	"/usr/share/john"
#define JOHN_PRIVATE_HOME	"~/.john"
#endif

/*
 * Crash recovery file format version strings.
 */
#define RECOVERY_VERSION_0	"REC0"
#define RECOVERY_VERSION_1	"REC1"
#define RECOVERY_VERSION_2	"REC2"
#define RECOVERY_VERSION_CURRENT	RECOVERY_VERSION_2

/*
 * Charset file format version string.
 */
#define CHARSET_VERSION		"CHR1"
```

```
/*
 * Timer interval in seconds.
 */
#define TIMER_INTERVAL            1

/*
 * Default crash recovery file saving delay in timer intervals.
 */
#define TIMER_SAVE_DELAY          (600 / TIMER_INTERVAL)

/*
 * Benchmark time in seconds, per cracking algorithm.
 */
#define BENCHMARK_TIME            5

/*
 * Number of salts to assume when benchmarking.
 */
#define BENCHMARK_MANY            0x100

/*
 * File names.
 */
#define CFG_FULL_NAME             "$JOHN/pwd_ldr.conf"
#define CFG_ALT_NAME              "$JOHN/pwd_ldr.ini"
#if JOHN_SYSTEMWIDE
#define CFG_PRIVATE_FULL_NAME           JOHN_PRIVATE_HOME
"/john.conf"
#define CFG_PRIVATE_ALT_NAME      JOHN_PRIVATE_HOME "/john.ini"
#define POT_NAME             JOHN_PRIVATE_HOME "/john.pot"
#define LOG_NAME             JOHN_PRIVATE_HOME "/john.log"
#define RECOVERY_NAME             JOHN_PRIVATE_HOME "/john.rec"
#else
#define POT_NAME             "$JOHN/pwd_ldr.pot"
#define LOG_NAME             "$JOHN/pwd_ldr.log"
#define RECOVERY_NAME             "$JOHN/pwd_ldr.rec"
#endif
#define LOG_SUFFIX                ."log"
#define RECOVERY_SUFFIX                ."rec"
#define WORDLIST_NAME             "$JOHN/password.lst"

//BOINC Add parameters
#define DB_NAME                   "john_db"
#define WU_TEMPLATE               "john_wu.xml"
#define RE_TEMPLATE               "john_re.xml"
#define DEFAULT_PRIORITY     20
#define DEF_APP_NAME         "john_boinc"
#define BATCH_NOT_SET       -1
//Batch flags
#define SINGLE_MODE               1
#define WORDLIST_MODE        2
#define INCREMENTAL_MODE     3
#define COMPLETE            4
#define ERROR              -1
```

```
//Status Flags
#define NOT_WORKED          1
#define WORKING             2
#define CRACKED             3
#define ERROR               -1
/*
 * Configuration file section names.
 */
#define SECTION_OPTIONS                 "Options"
#define SECTION_RULES             "List.Rules:"
#define SUBSECTION_SINGLE         "Single"
#define SUBSECTION_WORDLIST       "Wordlist"
#define SECTION_INC               "Incremental:"
#define SECTION_EXT               "List.External:"
#define SECTION_BOINC             "Boinc"
/*
 *  Hash  table  sizes.  These  are  also  hardcoded  into  the  hash
functions.
 */
#define SALT_HASH_SIZE            0x400
#define PASSWORD_HASH_SIZE_0      0x10
#define PASSWORD_HASH_SIZE_1      0x100
#define PASSWORD_HASH_SIZE_2      0x1000

/*
 * Password hash table thresholds. These are the counts of entries
required
 * to enable the corresponding hash table size.
 */
#define PASSWORD_HASH_THRESHOLD_0 (PASSWORD_HASH_SIZE_0 / 2)
#define PASSWORD_HASH_THRESHOLD_1 (PASSWORD_HASH_SIZE_1 / 4)
#define PASSWORD_HASH_THRESHOLD_2 (PASSWORD_HASH_SIZE_2 / 4)

/*
 * Tables of the above values.
 */
extern int password_hash_sizes[3];
extern int password_hash_thresholds[3];

/*
 * Cracked password hash size, used while loading.
 */
#define CRACKED_HASH_LOG          10
#define CRACKED_HASH_SIZE         (1 << CRACKED_HASH_LOG)

/*
 * Password hash function to use while loading.
 */
#define LDR_HASH_SIZE  (PASSWORD_HASH_SIZE_2     *     sizeof(struct
db_password *))
#define LDR_HASH_FUNC  (format->methods.binary_hash[2])

/*
 * Buffered keys hash size, used for "single crack" mode.
```

```
 */
#define SINGLE_HASH_LOG                5
#define SINGLE_HASH_SIZE           (1 << SINGLE_HASH_LOG)

/*
 * Minimum buffered keys hash size, used if min_keys_per_crypt is
even less.
 */
#define SINGLE_HASH_MIN                8

/*
 * Shadow file entry table hash size, used by unshadow.
 */
#define SHADOW_HASH_LOG                8
#define SHADOW_HASH_SIZE           (1 << SHADOW_HASH_LOG)

/*
 * Hash and buffer sizes for unique.
 */
#define UNIQUE_HASH_LOG                17
#define UNIQUE_HASH_SIZE           (1 << UNIQUE_HASH_LOG)
#define UNIQUE_BUFFER_SIZE         0x800000

/*
 * Maximum number of GECOS words per password to load.
 */
#define LDR_WORDS_MAX              0x10

/*
 * Maximum number of GECOS words to try in pairs.
 */
#define SINGLE_WORDS_PAIR_MAX          4

/*
 * Charset parameters.
 * Be careful if you change these, ((SIZE ** LENGTH) * SCALE) should
fit
 * into 64 bits. You can reduce the SCALE if required.
 */
#define CHARSET_MIN                ' '
#define CHARSET_MAX                0x7E
#define CHARSET_SIZE               (CHARSET_MAX - CHARSET_MIN + 1)
#define CHARSET_LENGTH             8
#define CHARSET_SCALE              0x100

/*
 * Compiler parameters.
 */
#define C_TOKEN_SIZE               0x100
#define C_UNGET_SIZE               (C_TOKEN_SIZE + 4)
#define C_EXPR_SIZE                0x100
#define C_STACK_SIZE               ((C_EXPR_SIZE + 4) * 4)
#define C_ARRAY_SIZE               0x1000000
#define C_DATA_SIZE                0x8000000
```

```
/*
 * Buffer size for rules.
 */
#define RULE_BUFFER_SIZE          0x100

/*
 * Maximum number of character ranges for rules.
 */
#define RULE_RANGES_MAX                8

/*
 * Buffer size for words while applying rules, should be at least as
large
 * as PLAINTEXT_BUFFER_SIZE.
 */
#define RULE_WORD_SIZE            0x80

/*
 * Buffer size for plaintext passwords.
 */
#define PLAINTEXT_BUFFER_SIZE          0x80

/*
 * Buffer size for fgets().
 */
#define LINE_BUFFER_SIZE          0x400

/*
 * john.pot and log file buffer sizes, can be zero.
 */
#define POT_BUFFER_SIZE                0x1000
#define LOG_BUFFER_SIZE                0x1000

/*
 * Buffer size for path names.
 */
#ifdef PATH_MAX
#define PATH_BUFFER_SIZE          PATH_MAX
#else
#define PATH_BUFFER_SIZE          0x400
#endif

#endif
```

THIS PAGE INTENTIONALLY LEFT BLANK

APPENDIX C: CLIENT APPLICATION

A. JOHN.C

```
/*************************************************************
 *  Distributed BOINC Password Cracker- Using John The Ripper
 *
 *  This is a BOINC app that cracks passwords.
 *
 *  7 June 2005 to use newest BOINC API as of vers. 4.52
 *
 *  John Crumpacker <jrcrumpa@nps.edu>  -  7 June 2005
 *    Department of Computer Science, Naval Postgraduate School,
Monterey, California
 *  @(#) $Version: 4.57$

*************************************************************
***/
/*
 * This file is part of John the Ripper password cracker,
 * Copyright (c) 1996-2004 by Solar Designer
 * Modified for BOINC by John Crumpacker
 */

//  Stuff we only need on Windows:

#ifdef __CYGWIN32__
#  include "boinc_win.h"
#endif


// BOINC API

#include <boinc_api.h>       // boinc_init(),boinc_finish()
#include <filesys.h>         // boinc_fopen(), etc...
//#include <util.h>   // parse_command_line(), boinc_sleep()
#include <diagnostics.h>     // boinc_init_diagnostics()




// C lib
#include <stdio.h>
#include <unistd.h>
#include <string.h>
#include <stdlib.h>
#include <sys/stat.h>

// John the ripper includes
#include "arch.h"
#include "misc.h"
```

```
#include "params.h"
#include "path.h"
#include "memory.h"
#include "list.h"
#include "tty.h"
#include "signals.h"
#include "common.h"
#include "formats.h"
#include "loader.h"
#include "logger.h"
#include "status.h"
#include "options.h"
#include "config.h"
#include "bench.h"
#include "charset.h"
#include "single.h"
#include "wordlist.h"
#include "inc.h"
#include "external.h"
#include "batch.h"
#include "cracker.h"

#if CPU_DETECT
extern int CPU_detect(void);
#endif

extern struct fmt_main fmt_DES, fmt_BSDI, fmt_MD5, fmt_BF;
extern struct fmt_main fmt_AFS, fmt_LM;

extern int unshadow(int argc, char **argv);
extern int unafs(int argc, char **argv);
extern int unique(int argc, char **argv);

static struct db_main database;
static struct fmt_main dummy_format;

static void john_register_one(struct fmt_main *format)
{
      if (options.format)
      if (strcmp(options.format, format->params.label)) return;

      fmt_register(format);
}

static void john_register_all(void)
{
      if (options.format) strlwr(options.format);

      john_register_one(&fmt_DES);
      john_register_one(&fmt_BSDI);
      john_register_one(&fmt_MD5);
      john_register_one(&fmt_BF);
      john_register_one(&fmt_AFS);
      john_register_one(&fmt_LM);
```

```
	if (!fmt_list) {
		fprintf(stderr, "Unknown ciphertext format name
requested\n");
		error();
	}
}

static void john_log_format(void)
{
	int min_chunk, chunk;

	log_event("- Hash type: %.100s (lengths up to %d%s)",
		database.format->params.format_name,
		database.format->params.plaintext_length,
		database.format->methods.split != fmt_default_split ?
		", longer passwords split" : "");

	log_event("- Algorithm: %.100s",
		database.format->params.algorithm_name);

	chunk = min_chunk = database.format-
>params.max_keys_per_crypt;
	if (options.flags & (FLG_SINGLE_CHK | FLG_BATCH_CHK) &&
	    chunk < SINGLE_HASH_MIN)
		chunk = SINGLE_HASH_MIN;
	if (chunk > 1)
		log_event("- Candidate passwords %s be buffered and "
			"tried in chunks of %d",
			min_chunk > 1 ? "will" : "may",
			chunk);
}

static char *john_loaded_counts(void)
{
	static char s_loaded_counts[80];

	if (database.password_count == 1)
		return "1 password hash";

	sprintf(s_loaded_counts,
		database.salt_count > 1 ?
		"%d password hashes with %d different salts" :
		"%d password hashes with no different salts",
		database.password_count,
		database.salt_count);

	return s_loaded_counts;
}

static void john_load(void)
{
	struct list_entry *current;
```

```
	umask(077);

/*	if (options.flags & FLG_EXTERNAL_CHK)
		ext_init(options.external);

	if (options.flags & FLG_MAKECHR_CHK) {
		options.loader.flags |= DB_CRACKED;
		ldr_init_database(&database, &options.loader);

		if (options.flags & FLG_PASSWD) {
			ldr_show_pot_file(&database, POT_NAME);

			database.options->flags |= DB_PLAINTEXTS;
			if ((current = options.passwd->head))
			do {
				ldr_show_pw_file(&database, current->data);
			} while ((current = current->next));
		} else {
			database.options->flags |= DB_PLAINTEXTS;
			ldr_show_pot_file(&database, POT_NAME);
		}

		return;
	}

	if (options.flags & FLG_STDOUT) {
		ldr_init_database(&database, &options.loader);
		database.format = &dummy_format;
		memset(&dummy_format, 0, sizeof(dummy_format));
		dummy_format.params.plaintext_length = options.length;
		dummy_format.params.flags = FMT_CASE | FMT_8_BIT;
	}
*/
	if (options.flags & FLG_PASSWD) {
/*		if (options.flags & FLG_SHOW_CHK) {
			options.loader.flags |= DB_CRACKED;
			ldr_init_database(&database, &options.loader);

			ldr_show_pot_file(&database, POT_NAME);

			if ((current = options.passwd->head))
			do {
				ldr_show_pw_file(&database, current->data);
			} while ((current = current->next));

			printf("%s%d password hash%s cracked, %d left\n",
				database.guess_count ? "\n" : "",
				database.guess_count,
				database.guess_count != 1 ? "es" : "",
				database.password_count -
				database.guess_count);

			return;
		}
```

```
*/
            if (options.flags & (FLG_SINGLE_CHK | FLG_BATCH_CHK)){
                  options.loader.flags |= DB_WORDS;
                  fprintf(stderr,"            Loader            flags:
%d\n",options.loader.flags);
            }

/*          if (mem_saving_level)
                  options.loader.flags &= ~DB_LOGIN;
*/          ldr_init_database(&database, &options.loader);

            if ((current = options.passwd->head))
            do {
                  fprintf(stderr,"Password    file:    %s\n",current-
>data);
                  ldr_load_pw_file(&database, current->data);
            } while ((current = current->next));

            if ((options.flags & FLG_CRACKING_CHK) &&
                database.password_count) {
//                log_init(LOG_NAME, NULL, options.session);
/*                if (status_restored_time)
                        log_event("Continuing      an      interrupted
session");
                  else
*/                      log_event("Starting a new session");
                  log_event("Loaded      a      total      of      %s",
john_loaded_counts());
            }

//          ldr_load_pot_file(&database, POT_NAME);

            ldr_fix_database(&database);

            if (database.password_count) {
                  log_event("Remaining %s", john_loaded_counts());
                  printf("Loaded %s (%s [%s])\n",
                        john_loaded_counts(),
                        database.format->params.format_name,
                        database.format->params.algorithm_name);
            } else {
                  log_discard();
                  puts("No password hashes loaded");
            }

            if  ((options.flags  &  FLG_PWD_REQ)  &&  !database.salts)
exit(0);
      }
}

static void john_init(int argc, char **argv)
{
#if CPU_DETECT
      if (!CPU_detect()) {
```

```
#if CPU_REQ
#if CPU_FALLBACK
#if defined(__DJGPP__) || defined(__CYGWIN32__)
#error CPU_FALLBACK is incompatible with the current DOS and Win32
code
#endif
            execv(JOHN_SYSTEMWIDE_EXEC    "/"    CPU_FALLBACK_BINARY,
argv);
            perror("execv");
#endif
            fprintf(stderr, "Sorry, %s is required\n", CPU_NAME);
            error();
#endif
      }
#endif
      char res_name_conf[256];
      char res_name_cr[256];
      char res_name_pw[256];
      path_init(argv);
      fprintf(stderr,"Before any cfg_init call\n");
/*#if JOHN_SYSTEMWIDE
      cfg_init(CFG_PRIVATE_FULL_NAME, 1);
      cfg_init(CFG_PRIVATE_ALT_NAME, 1);
#endif
*/
      if
(boinc_resolve_filename(CFG_FULL_NAME,res_name_conf,sizeof(res_name_
conf)))
            fprintf(stderr,"Could not resolve %s\n",CFG_FULL_NAME);


      cfg_init(res_name_conf, 1,0);
//    cfg_init(CFG_ALT_NAME, 0,0);
      //Resolves the Physical name of crdata
      fprintf(stderr,"before Crack data boinc_resolve\n");


      if
(boinc_resolve_filename("crdata",res_name_cr,sizeof(res_name_cr)))
            fprintf(stderr,"Could not resolve crdata\n");
      fprintf(stderr,"after boinc_resolve\n");
      cfg_init(res_name_cr,0,1);

      status_init(NULL, 1);
      if
(boinc_resolve_filename("passwdfile",res_name_pw,sizeof(res_name_pw)
))
            fprintf(stderr,"Count not resolve passwordfile\n");
      fprintf(stderr,"Before opt_init:\n");
      opt_init(argc, argv, res_name_pw);

      john_register_all();
      common_init();
```

```
	sig_init();

	john_load();
}

static void john_run(void)
{
/*	if (options.flags & FLG_TEST_CHK)
		benchmark_all();
	else
	if (options.flags & FLG_MAKECHR_CHK)
		do_makechars(&database, options.charset);
	else*/
	if (options.flags & FLG_CRACKING_CHK) {
	/*	if (!(options.flags & FLG_STDOUT)) {
			status_init(NULL, 1);
			log_init(LOG_NAME, POT_NAME, options.session);
			john_log_format();
			if (cfg_get_bool(SECTION_OPTIONS, NULL, "Idle"))
				log_event("- Configured to use otherwise idle
"
					"processor cycles only");
		}*/
	//	tty_init();

		if (options.flags & FLG_SINGLE_CHK)
			do_single_crack(&database);
		else
		if (options.flags & FLG_WORDLIST_CHK)
			do_wordlist_crack(&database, options.wordlist,
				(options.flags & FLG_RULES) != 0);
		else
		if (options.flags & FLG_INC_CHK)
			do_incremental_crack(&database, options.charset);
/*		else
		if (options.flags & FLG_EXTERNAL_CHK)
			do_external_crack(&database);
		else
		if (options.flags & FLG_BATCH_CHK)
			do_batch_crack(&database);
*/
//Might not want this going to the screen
	//	status_print();
	//	tty_done();
	}
}

static void john_done(void)
{
	int retval;
	char res_name[254];
	FILE *out;
	fprintf(stderr,"Inside john_done\n");
```

```
      retval = boinc_resolve_filename(OUTPUT_FILENAME,res_name
                                      ,sizeof(res_name));
      fprintf(stderr,"output res_name:%s\n",res_name);
      if (!(out = boinc_fopen(res_name, "w")))
            pexit("boinc_fopen: %s",res_name);

      print_crk_db(out);
      fprintf(stderr,"After print_crk_db\n");

      if (fclose(out)) error();

      path_done();

      if ((options.flags & FLG_CRACKING_CHK) &&
          !(options.flags & FLG_STDOUT)) {
            if (event_abort)
                  log_event("Session aborted");
            else
                  log_event("Session completed");
      }
      log_done();

      check_abort(0);
}

int main(int argc, char **argv)
{
      int rc;

      char *name;


#ifdef __DJGPP__
      if (--argc <= 0) return 1;
      if ((name = strrchr(argv[0], '/')))
            strcpy(name + 1, argv[1]);
      name = argv[1];
      argv[1] = argv[0];
      argv++;
#else
      if (!argv[0])
            name = "";
      else
      if ((name = strrchr(argv[0], '/')))
            name++;
      else
            name = argv[0];
#endif

#ifdef __CYGWIN32__
      if (strlen(name) > 4)
      if (!strcmp(strlwr(name) + strlen(name) - 4, .”exe"))
            name[strlen(name) - 4] = 0;
#endif
```

```
      if (!strcmp(name, "unshadow"))
            return unshadow(argc, argv);

      if (!strcmp(name, "unafs"))
            return unafs(argc, argv);

      if (!strcmp(name, "unique"))
            return unique(argc, argv);
      fprintf(stderr,"boinc diag\n");
      rc = boinc_init_diagnostics(BOINC_DIAG_REDIRECTSTDERR);
      if(rc) exit(rc);
      fprintf(stderr,"boinc init\n");

      rc = boinc_init();
      if (rc){
        fprintf(stderr, "APP: boinc_init() failed.\n");
        exit(rc);
      }

      john_init(argc, argv);
      john_run();
      john_done();
      boinc_finish(rc);    /* back to BOINC core */

      return 0;
}


/*  Dummy graphics API entry points.
 *   This app does not do graphics, but it still must provide these
callbacks.
 */

/*void app_graphics_init() {}
void app_graphics_resize(int width, int height){}
void app_graphics_render(int xs, int ys, double time_of_day) {}
void app_graphics_reread_prefs() {}
void boinc_app_mouse_move(int x, int y, bool left, bool middle, bool
right ){}
void boinc_app_mouse_button(int x, int y, int which, bool is_down){}
void boinc_app_key_press(int, int){}
void boinc_app_key_release(int, int){}
*/
```

B. BOINC.H

```
extern int resolve_filename(char *, char *, int);
```

C. BOINC.C

```
/*
 * This File contains the wrappers for all the C++ header functions
 */

#ifdef __CYGWIN32__
#  include "boinc_win.h"
#endif


// BOINC API

#include <boinc_api.h>       // boinc_init(),boinc_finish()
#include <filesys.h>         // boinc_fopen(), etc...
#include <util.h>   // parse_command_line(), boinc_sleep()
#include <diagnostics.h>     // boinc_init_diagnostics()



int resolve_filename( char *logic, char *res_name, int size){
      return boinc_resolve_filename(logic,res_name,size);
}
```

D. CRACKER.H

```
/*
 * This file is part of John the Ripper password cracker,
 * Copyright (c) 1996-99 by Solar Designer
 */

/*
 * Cracking routines.
 */

#ifndef _JOHN_CRACKER_H
#define _JOHN_CRACKER_H

#include "loader.h"

/*
 * Initializes the cracker for a password database (should not be
empty).
 * If fix_state() is not NULL, it will be called when key buffer
becomes
 * empty, its purpose is to save current state for possible recovery
in
 * the future. If guesses is not NULL, the cracker will save guessed
keys
 * in there (the caller must make sure there's room).
 */
extern void crk_init(struct db_main *db, void (*fix_state)(void),
```

```
      struct db_keys *guesses);

/*
 * Tries the key against all passwords in the database (should not
be empty).
 * The return value is non-zero if aborted or everything got cracked
(event
 * flags can be used to find out which of these has happened).
 */
extern int crk_process_key(char *key);

/*
 * Resets the guessed keys buffer and processes all the buffered
keys for
 * this salt. The return value is the same as for crk_process_key().
 */
extern int crk_process_salt(struct db_salt *salt);

/*
 * Return current keys range, crk_get_key2() may return NULL if
there's only
 * one key. Note: these functions may share a static result buffer.
 */
extern char *crk_get_key1(void);
extern char *crk_get_key2(void);

/*
 * Processes all the buffered keys (unless aborted).
 */
extern void crk_done(void);
/*
 * Add passwords to the crack DB used for tracking
 * program output.
 */
extern void add_crk_pw(char *cipher_t, int pw_id);

/*
 * Print cracker database to file
 */
extern void print_crk_db(FILE *F);

extern struct db_cracked *crack_db;

#endif
```

E. CRACKER.C

```
/*
 * This file is part of John the Ripper password cracker,
 * Copyright (c) 1996-2003 by Solar Designer
 */

#include <string.h>
```

```
#include "arch.h"
#include "misc.h"
#include "math.h"
#include "params.h"
#include "memory.h"
#include "signals.h"
#include "idle.h"
#include "formats.h"
#include "loader.h"
#include "logger.h"
#include "status.h"
#include "recovery.h"

#ifdef index
#undef index
#endif

static struct db_main *crk_db;
static struct fmt_params crk_params;
static struct fmt_methods crk_methods;
static int crk_key_index, crk_last_key;
static void *crk_last_salt;
static void (*crk_fix_state)(void);
static struct db_keys *crk_guesses;
static int64 *crk_timestamps;
static char crk_stdout_key[PLAINTEXT_BUFFER_SIZE];
static struct db_cracked *crack_db= NULL;
static void crk_dummy_set_salt(void *salt)
{
}

static void crk_dummy_fix_state(void)
{
}

static void crk_init_salt(void)
{
      if (!crk_db->salts->next) {
            crk_methods.set_salt(crk_db->salts->salt);
            crk_methods.set_salt = crk_dummy_set_salt;
      }
}

void add_crk_pw( char *cipher_t, int pw_id){

      struct db_cracked *last;
      fprintf(stderr,"Inside Add_crk_pw\n");
      last = crack_db;
      crack_db = mem_alloc_tiny(sizeof(struct db_cracked),
                        MEM_ALIGN_WORD);
      crack_db->next=last;
      crack_db->plaintext=NULL;
      crack_db->ciphertext=str_alloc_copy(cipher_t);
```

```
	crack_db->pwid=pw_id;
	fprintf(stderr,"pwid: %d\nCiphertext: %s\n",crack_db->pwid
			,crack_db->ciphertext);
}

void update_crk_pw(char *cipher_t, char *plain_t, int pw_id){
	struct db_cracked *ptr;
	fprintf(stderr,"Inside update_crk_pw\n");
	ptr = crack_db;

	do{
		if ((pw_id==ptr->pwid)
			&& (!strcmp(ptr->ciphertext, cipher_t))){
			ptr->plaintext=str_alloc_copy(plain_t);
			fprintf(stderr,"Found Password:%d:%s:%s\n",pw_id
					,cipher_t,plain_t);
			return;
		}

	}while ( ((ptr = ptr->next)!= NULL) );

}
void print_crk_db(FILE *f){
	struct db_cracked *ptr;
	ptr=crack_db;
	fprintf(stderr,"Inside print_crk_db\n");
	fprintf(stderr,"first pwid: %d\n",ptr->pwid);
	do{
//		fprintf(stderr,"before plaintext ptr check\n");
		if (!ptr->plaintext){
//			fprintf(stderr," before str_alloc_copy\n");
			ptr->plaintext=str_alloc_copy("NO_PASSWORD");
		}
//		fprintf(stderr,"ptr->plaintext: %s\n",ptr->plaintext);
		fprintf(f,"%d:%s:%s\n",ptr->pwid,ptr->ciphertext,
				ptr->plaintext);
		fprintf(stderr,"%d:%s:%s\n",ptr->pwid,ptr->ciphertext,
				ptr->plaintext);
	}while( ((ptr=ptr->next)!= NULL) );

}

void crk_init(struct db_main *db, void (*fix_state)(void),
	struct db_keys *guesses)
{
	char *where;
	size_t size;



	if (db->loaded)
	if ((where = fmt_self_test(db->format))) {
		log_event("! Self test failed (%s)", where);
		fprintf(stderr, "Self test failed (%s)\n", where);
```

```
            error();
      }

      crk_db = db;
      memcpy(&crk_params,     &db->format->params,     sizeof(struct
fmt_params));
      memcpy(&crk_methods,    &db->format->methods,    sizeof(struct
fmt_methods));

      if (db->loaded) crk_init_salt();
      crk_last_key = crk_key_index = 0;
      crk_last_salt = NULL;

      if (fix_state)
            (crk_fix_state = fix_state)();
      else
            crk_fix_state = crk_dummy_fix_state;

      crk_guesses = guesses;

      if (db->loaded) {
            size = crk_params.max_keys_per_crypt * sizeof(int64);
            memset(crk_timestamps = mem_alloc(size), -1, size);
      } else
            crk_stdout_key[0] = 0;

      rec_save();

      idle_init();
}

static   int   crk_process_guess(struct   db_salt   *salt,   struct
db_password *pw,
      int index)
{
      int dupe;
      char *key;
      struct db_salt *search_salt;
      struct db_password *search_pw;

      fprintf(stderr,"Inside crk_process_guess\n");
      dupe    =    !memcmp(&crk_timestamps[index],    &status.crypts,
sizeof(int64));
      crk_timestamps[index] = status.crypts;

      key = crk_methods.get_key(index);
/* here is where I need to add a file to print...
   Need a structure that will handle this ...
   that means I can make a list of cracked passwords
   which allows me  */
      fprintf(stderr,"before update_crk_pw\n");
      update_crk_pw(pw->source,key,pw->pwid);
//    log_guess(crk_db->options->flags & DB_LOGIN ? pw->login : "?",
//          dupe ? NULL : pw->source, key);
```

```
	fprintf(stderr,"after update_crk_pw\n");

	crk_db->password_count--;
	crk_db->guess_count++;
	status.guess_count++;

	if (crk_guesses && !dupe) {
		strnfcpy(crk_guesses->ptr,                                key,
crk_params.plaintext_length);
		crk_guesses->ptr += crk_params.plaintext_length;
		crk_guesses->count++;
	}

	if (pw == salt->list) {
		salt->list = pw->next;

		ldr_update_salt(crk_db, salt);

		if (!salt->list) {
			crk_db->salt_count--;

			if (salt == crk_db->salts) {
				crk_db->salts = salt->next;
			} else {
				search_salt = crk_db->salts;
				while (search_salt->next != salt)
					search_salt = search_salt->next;
				search_salt->next = salt->next;
			}

			if (crk_db->salts) crk_init_salt(); else return 1;
		}
	} else {
		search_pw = salt->list;
		while (search_pw->next != pw)
			search_pw = search_pw->next;
		search_pw->next = pw->next;

		ldr_update_salt(crk_db, salt);
	}

	return 0;
}

static int crk_process_event(void)
{
	event_pending = 0;

	if (event_save) {
		event_save = 0;
		rec_save();
	}

	if (event_status) {
```

```
            event_status = 0;
            status_print();
      }

      return event_abort;
}

static int crk_password_loop(struct db_salt *salt)
{
      struct db_password *pw;
      int index;

#if !OS_TIMER
      sig_timer_emu_tick();
#endif

      idle_yield();

      if (event_pending)
      if (crk_process_event()) return 1;

      crk_methods.crypt_all(crk_key_index);

      status_update_crypts(salt->count * crk_key_index);

      if (salt->hash_size < 0) {
            pw = salt->list;
            do {
                  if (crk_methods.cmp_all(pw->binary, crk_key_index))
                  for (index = 0; index < crk_key_index; index++)
                  if (crk_methods.cmp_one(pw->binary, index))
                  if (crk_methods.cmp_exact(pw->source, index)) {
                        if (crk_process_guess(salt, pw, index))
                              return 1;
                        else
                              break;
                  }
            } while ((pw = pw->next));
      } else
      for (index = 0; index < crk_key_index; index++) {
            if ((pw = salt->hash[salt->index(index)]))
            do {
                  if (crk_methods.cmp_one(pw->binary, index))
                  if (crk_methods.cmp_exact(pw->source, index))
                  if (crk_process_guess(salt, pw, index))
                        return 1;
            } while ((pw = pw->next_hash));
      }

      return 0;
}

static int crk_salt_loop(void)
{
```

```
	struct db_salt *salt;

	salt = crk_db->salts;
	do {
		crk_methods.set_salt(salt->salt);
		if (crk_password_loop(salt)) return 1;
	} while ((salt = salt->next));

	crk_last_key = crk_key_index; crk_key_index = 0;
	crk_last_salt = NULL;
	crk_fix_state();

	crk_methods.clear_keys();

	return 0;
}

int crk_process_key(char *key)
{
	if (crk_db->loaded) {
		crk_methods.set_key(key, crk_key_index++);

		if (crk_key_index >= crk_params.max_keys_per_crypt)
			return crk_salt_loop();

		return 0;
	}

#if !OS_TIMER
	sig_timer_emu_tick();
#endif

	if (event_pending)
	if (crk_process_event()) return 1;

	puts(strnzcpy(crk_stdout_key, key, crk_params.plaintext_length
+ 1));

	status_update_crypts(1);
	crk_fix_state();

	return 0;
}

int crk_process_salt(struct db_salt *salt)
{
	char *ptr;
	char key[PLAINTEXT_BUFFER_SIZE];
	int count, index;

	if (crk_guesses) {
		crk_guesses->count = 0;
		crk_guesses->ptr = crk_guesses->buffer;
	}
```

```
	if (crk_last_salt != salt->salt)
		crk_methods.set_salt(crk_last_salt = salt->salt);

	ptr = salt->keys->buffer;
	count = salt->keys->count;
	index = 0;

	crk_methods.clear_keys();

	while (count--) {
		strnzcpy(key, ptr, crk_params.plaintext_length + 1);
		ptr += crk_params.plaintext_length;

		crk_methods.set_key(key, index++);
		if (index >= crk_params.max_keys_per_crypt || !count) {
			crk_key_index = index;
			if (crk_password_loop(salt)) return 1;
			if (!salt->list) return 0;
			index = 0;
		}
	}

	return 0;
}

char *crk_get_key1(void)
{
	if (crk_db->loaded)
		return crk_methods.get_key(0);
	else
		return crk_stdout_key;
}

char *crk_get_key2(void)
{
	if (crk_key_index > 1)
		return crk_methods.get_key(crk_key_index - 1);
	else
	if (crk_last_key > 1)
		return crk_methods.get_key(crk_last_key - 1);
	else
		return NULL;
}

void crk_done(void)
{
	if (crk_db->loaded) {
		if (crk_key_index && crk_db->salts && !event_abort)
			crk_salt_loop();

		MEM_FREE(crk_timestamps);
	}
}
```

F. INC.H

```
/*
 * This file is part of John the Ripper password cracker,
 * Copyright (c) 1996-98 by Solar Designer
 */

/*
 * Incremental mode cracker.
 */

#ifndef _JOHN_INC_H
#define _JOHN_INC_H

#include "loader.h"

/*
 * Runs the incremental mode cracker.
 */
extern void do_incremental_crack(struct db_main *db, char *mode);

#endif
```

G. INC.C

```
/*
 * This file is part of John the Ripper password cracker,
 * Copyright (c) 1996-2004 by Solar Designer
 */

#include <stdio.h>
#include <string.h>
#include <filesys.h>
#include <boinc_api.h>

#include "arch.h"
#include "misc.h"
#include "params.h"
#include "path.h"
#include "memory.h"
#include "signals.h"
#include "formats.h"
#include "loader.h"
#include "logger.h"
#include "status.h"
#include "recovery.h"
#include "config.h"
#include "charset.h"
#include "external.h"
#include "cracker.h"
```

```
struct TRACKER{
    int s_loc;
    int s_length;
    int s_fixed;
    int s_count;
    int s_num0;
    int s_num1;
    int s_num2;
    int s_num3;
    int s_num4;
    int s_num5;
    int s_num6;
    int s_num7;
    int f_loc;
    int f_length;
    int f_fixed;
    int f_count;
    int f_num0;
    int f_num1;
    int f_num2;
    int f_num3;
    int f_num4;
    int f_num5;
    int f_num6;
    int f_num7;
};

extern struct fmt_main fmt_LM;

typedef char (*char2_table)
      [CHARSET_SIZE + 1][CHARSET_SIZE + 1];
typedef char (*chars_table)
      [CHARSET_SIZE + 1][CHARSET_SIZE + 1][CHARSET_SIZE + 1];

static int rec_compat;
static int rec_entry;
static int rec_numbers[CHARSET_LENGTH];

static int entry;
static int numbers[CHARSET_LENGTH];

static void save_state(FILE *file)
{
      int pos;

      fprintf(file,     "%d\n%d\n%d\n",     rec_entry,     rec_compat,
CHARSET_LENGTH);
      for (pos = 0; pos < CHARSET_LENGTH; pos++)
            fprintf(file, "%d\n", rec_numbers[pos]);
}

static int restore_state(FILE *file)
{
      int length;
```

```
	int pos;

	if (fscanf(file, "%d\n", &rec_entry) != 1) return 1;
	rec_compat = 1;
	length = CHARSET_LENGTH;
	if (rec_version >= 2) {
		if (fscanf(file, "%d\n%d\n", &rec_compat, &length) != 2)
			return 1;
		if ((unsigned int)rec_compat > 1) return 1;
		if ((unsigned int)length > CHARSET_LENGTH) return 1;
	}
	for (pos = 0; pos < length; pos++) {
		if (fscanf(file, "%d\n", &rec_numbers[pos]) != 1) return
1;
		if  ((unsigned  int)rec_numbers[pos]  >=  CHARSET_SIZE)
return 1;
	}

	return 0;
}

static void fix_state(void)
{
	rec_entry = entry;
	memcpy(rec_numbers, numbers, sizeof(rec_numbers));
}

static void inc_format_error(char *charset)
{
	log_event("! Incorrect charset file format: %.100s", charset);
	fprintf(stderr,  "Incorrect  charset  file  format:  %s\n",
charset);
	error();
}

static void inc_new_length(unsigned int length,
	struct charset_header *header, FILE *file, char *charset,
	char *char1, char2_table char2, chars_table *chars)
{
	long offset;
	int value, pos, i, j;
	char *buffer;
	int count;

	log_event("- Switching to length %d", length + 1);

	char1[0] = 0;
	if (length)
		memset(char2, 0, sizeof(*char2));
	for (pos = 0; pos <= (int)length - 2; pos++)
		memset(chars[pos], 0, sizeof(**chars));

	offset =
		(long)header->offsets[length][0] +
```

```
      ((long)header->offsets[length][1] << 8) +
      ((long)header->offsets[length][2] << 16) +
      ((long)header->offsets[length][3] << 24);
if (fseek(file, offset, SEEK_SET)) pexit("fseek");

i = j = pos = -1;
if ((value = getc(file)) != EOF)
do {
      if (value != CHARSET_ESC) {
            switch (pos) {
            case -1:
                  inc_format_error(charset);

            case 0:
                  buffer = char1;
                  break;

            case 1:
                  if (j < 0)
                        inc_format_error(charset);
                  buffer = (*char2)[j];
                  break;

            default:
                  if (i < 0 || j < 0)
                        inc_format_error(charset);
                  buffer = (*chars[pos - 2])[i][j];
            }

            buffer[count = 0] = value;
            while ((value = getc(file)) != EOF) {
                  buffer[++count] = value;
                  if (value == CHARSET_ESC) break;
                  if (count >= CHARSET_SIZE)
                        inc_format_error(charset);
            }
            buffer[count] = 0;

            continue;
      }

      if ((value = getc(file)) == EOF) break; else
      if (value == CHARSET_NEW) {
            if ((value = getc(file)) != (int)length) break;
            if ((value = getc(file)) == EOF) break;
            if ((unsigned int)value > length)
                  inc_format_error(charset);
            pos = value;
      } else
      if (value == CHARSET_LINE) {
            if (pos < 0)
                  inc_format_error(charset);
            if ((value = getc(file)) == EOF) break;
            if ((unsigned int)(i = value) > CHARSET_SIZE)
```

```
                        inc_format_error(charset);
                  if ((value = getc(file)) == EOF) break;
                  if ((unsigned int)(j = value) > CHARSET_SIZE)
                        inc_format_error(charset);
            } else
                  inc_format_error(charset);

            value = getc(file);
      } while (value != EOF);

      if (value == EOF) {
            if (ferror(file))
                  pexit("getc");
            else
                  inc_format_error(charset);
      }
}

static void expand(char *dst, char *src, int size)
{
      char *dptr = dst, *sptr = src;
      int count = size;
      char present[CHARSET_SIZE];

      memset(present, 0, sizeof(present));
      while (*dptr) {
            if (--count <= 1) return;
            present[ARCH_INDEX(*dptr++) - CHARSET_MIN] = 1;
      }

      while (*sptr)
      if (!present[ARCH_INDEX(*sptr) - CHARSET_MIN]) {
            *dptr++ = *sptr++;
            if (--count <= 1) break;
      } else
            sptr++;
      *dptr = 0;
}

static void inc_new_count(unsigned int length, int count,
      char *allchars, char *char1, char2_table char2, chars_table
*chars)
{
      int pos, i, j;
      int size;

      log_event("- Expanding tables for length %d to character count
%d",
            length + 1, count + 1);

      size = count + 2;

      expand(char1, allchars, size);
      if (length)
```

```
            expand((*char2)[CHARSET_SIZE], allchars, size);
      for (pos = 0; pos <= (int)length - 2; pos++)
            expand((*chars[pos])[CHARSET_SIZE][CHARSET_SIZE],
                  allchars, size);

      for (i = 0; i < CHARSET_SIZE; i++) {
            if (length)
                  expand((*char2)[i], (*char2)[CHARSET_SIZE], size);

            for (j = 0; j < CHARSET_SIZE; j++)
            for (pos = 0; pos <= (int)length - 2; pos++) {
                  expand((*chars[pos])[i][j], (*chars[pos])
                        [CHARSET_SIZE][j], size);
                  expand((*chars[pos])[i][j], (*chars[pos])
                        [CHARSET_SIZE][CHARSET_SIZE], size);
            }
      }
}
int end_of_wu(struct TRACKER *t){
    return  (t->s_loc == t->f_loc) && (t->s_length == t->f_length)
        && (t->s_fixed == t->f_fixed) && (t->s_count == t->f_count)
        && (t->f_num0 == numbers[0]) && (t->f_num1 == numbers[1])
        && (t->f_num2 == numbers[2]) && (t->f_num3 == numbers[3])
        && (t->f_num4 == numbers[4]) && (t->f_num5 == numbers[5])
        && (t->f_num6 == numbers[6]) && (t->f_num7 == numbers[7]);

}

static int inc_key_loop(int length, int fixed, int count,
      char  *char1,  char2_table  char2,  chars_table  *chars,  struct
TRACKER *t)
{
      char key_i[PLAINTEXT_BUFFER_SIZE];
      char key_e[PLAINTEXT_BUFFER_SIZE];
      char *key;
      char *chars_cache;
      int numbers_cache;
      int pos;

      key_i[length + 1] = 0;
      numbers[fixed] = count;

      chars_cache = NULL;

update_all:
      pos = 0;
update_ending:
      if (pos < 2) {
            if (pos == 0)
                  key_i[0] = char1[numbers[0]];
            if (length)
                  key_i[1]        =        (*char2)[key_i[0]        -
CHARSET_MIN][numbers[1]];
            pos = 2;
```

```
	}
	while (pos < length) {
		key_i[pos] = (*chars[pos - 2])
			[ARCH_INDEX(key_i[pos - 2]) - CHARSET_MIN]
			[ARCH_INDEX(key_i[pos - 1]) - CHARSET_MIN]
			[numbers[pos]];
		pos++;
	}
	numbers_cache = numbers[length];
	if (pos == length) {
		chars_cache = (*chars[pos - 2])
			[ARCH_INDEX(key_i[pos - 2]) - CHARSET_MIN]
			[ARCH_INDEX(key_i[pos - 1]) - CHARSET_MIN];
update_last:
		key_i[length] = chars_cache[numbers_cache];
	}

	key = key_i;
	if (!ext_mode || !f_filter || ext_filter_body(key_i, key =
key_e)){
	    if (crk_process_key(key)) return 1;
	  if (end_of_wu(t)) return 1;
	}
	if (rec_compat) goto compat;

	pos = length;
	if (fixed < length) {
		if (++numbers_cache <= count) {
			if (length >= 2) goto update_last;
			numbers[length] = numbers_cache;
			goto update_ending;
		}
		numbers[pos--] = 0;
		while (pos > fixed) {
			if (++numbers[pos] <= count) goto update_ending;
			numbers[pos--] = 0;
		}
	}
	while (pos-- > 0) {
		if (++numbers[pos] < count) goto update_ending;
		numbers[pos] = 0;
	}

	return 0;

compat:
	pos = 0;
	if (fixed) {
		if (++numbers[0] < count) goto update_all;
		if (!length && numbers[0] <= count) goto update_all;
		numbers[0] = 0;
		pos = 1;
		while (pos < fixed) {
			if (++numbers[pos] < count) goto update_all;
```

```
                numbers[pos++] = 0;
          }
     }
     while (++pos <= length) {
          if (++numbers[pos] <= count) goto update_all;
          numbers[pos] = 0;
     }

     return 0;
}

void do_incremental_crack(struct db_main *db, char *mode)
{
     char *charset;
     int min_length, max_length, max_count;
     char *extra;
     FILE *file;
     struct charset_header *header;
     char allchars[CHARSET_SIZE + 1];
     char char1[CHARSET_SIZE + 1];
     char2_table char2;
     chars_table chars[CHARSET_LENGTH - 2];
     unsigned char *ptr;
     unsigned int length, fixed, count;
     unsigned int real_count;
     int last_length, last_count;
     int pos;
     char res_name[256];
    struct TRACKER *con;

    con = mem_alloc(sizeof(struct TRACKER));
     if (!mode) {
          if (db->format == &fmt_LM)
               mode = "LanMan";
          else
               mode = "All";
     }

     log_event("Proceeding  with  \"incremental\"  mode:  %.100s",
mode);

     if (!(charset = cfg_get_param(SECTION_INC, mode, "File"))) {
          log_event("! No charset defined");
          fprintf(stderr,  "No  charset  defined  for  mode:  %s\n",
mode);
          error();
     }

     extra = cfg_get_param(SECTION_INC, mode, "Extra");

     if ((min_length = cfg_get_int(SECTION_INC, mode, "MinLen")) <
0)
          min_length = 0;
```

```
	if ((max_length = cfg_get_int(SECTION_INC, mode, "MaxLen")) <
0)
		max_length = CHARSET_LENGTH;
	max_count = cfg_get_int(SECTION_INC, mode, "CharCount");

	if (min_length > max_length) {
		log_event("! MinLen = %d exceeds MaxLen = %d",
			min_length, max_length);
		fprintf(stderr, "MinLen = %d exceeds MaxLen = %d\n",
			min_length, max_length);
		error();
	}

	if (max_length > CHARSET_LENGTH) {
		log_event("! MaxLen = %d exceeds the compile-time limit
of %d",
			max_length, CHARSET_LENGTH);
		fprintf(stderr,
			"\n"
			"MaxLen = %d exceeds the compile-time limit of
%d\n\n"
			"There're several good reasons why you probably
don't "
			"need to raise it:\n"
			"- many hash types don't support passwords "
			"(or password halves) longer than\n"
			"7 or 8 characters;\n"
			"- you probably don't have sufficient statistical "
			"information to generate a\n"
			"charset file for lengths beyond 8;\n"
			"-  the  limitation  applies  to  incremental  mode
only.\n",
			max_length, CHARSET_LENGTH);
		error();
	}
	if          (boinc_resolve_filename(charset,          res_name,
sizeof(res_name)))
		fprintf(stderr,"Failed to resolve %s\n",charset);

	if (!(file = boinc_fopen(res_name, "rb")))
		pexit("fopen: %s", res_name);

	header = (struct charset_header *)mem_alloc(sizeof(*header));

	charset_read_header(file, header);
	if (ferror(file)) pexit("fread");

	if (feof(file) ||
	    memcmp(header->version,   CHARSET_VERSION,   sizeof(header-
>version)) ||
		header->min != CHARSET_MIN || header->max != CHARSET_MAX
||
		header->length != CHARSET_LENGTH ||
		header->count > CHARSET_SIZE || !header->count)
```

```
            inc_format_error(charset);

      fread(allchars, header->count, 1, file);
      if (ferror(file)) pexit("fread");
      if (feof(file)) inc_format_error(charset);

      allchars[header->count] = 0;
      if (extra)
            expand(allchars, extra, sizeof(allchars));
      real_count = strlen(allchars);

      if (max_count < 0) max_count = CHARSET_SIZE;

      if (min_length != max_length)
            log_event("-  Lengths  %d  to  %d,  up  to  %d  different
characters",
                  min_length, max_length, max_count);
      else
            log_event("- Length %d, up to %d different characters",
                  min_length, max_count);

      if ((unsigned int)max_count > real_count) {
            log_event("! Only %u characters available", real_count);
            fprintf(stderr,    "Warning:    only    %u    characters
available\n",
                  real_count);
      }

      if (header->length >= 2)
            char2 = (char2_table)mem_alloc(sizeof(*char2));
      else
            char2 = NULL;
      for (pos = 0; pos < (int)header->length - 2; pos++)
            chars[pos] = (chars_table)mem_alloc(sizeof(*chars[0]));

      rec_compat = 0;
      rec_entry = 0;
      memset(rec_numbers, 0, sizeof(rec_numbers));

      status_init(NULL, 0);

//    rec_restore_mode(restore_state);
//    rec_init(db, save_state);

    con->s_loc = cfg_get_int(SECTION_START_LOCATION,NULL,"s_loc");
    con->s_length                                                   =
cfg_get_int(SECTION_START_LOCATION,NULL,"s_length");
    con->s_fixed                                                    =
cfg_get_int(SECTION_START_LOCATION,NULL,"s_fixed");
    con->s_count                                                    =
cfg_get_int(SECTION_START_LOCATION,NULL,"s_count");
    con->s_num0 = cfg_get_int(SECTION_START_LOCATION,NULL,"s_num0");
    con->s_num1 = cfg_get_int(SECTION_START_LOCATION,NULL,"s_num1");
    con->s_num2 = cfg_get_int(SECTION_START_LOCATION,NULL,"s_num2");
```

```
    con->s_num3 = cfg_get_int(SECTION_START_LOCATION,NULL,"s_num3");
    con->s_num4 = cfg_get_int(SECTION_START_LOCATION,NULL,"s_num4");
    con->s_num5 = cfg_get_int(SECTION_START_LOCATION,NULL,"s_num5");
    con->s_num6 = cfg_get_int(SECTION_START_LOCATION,NULL,"s_num6");
    con->s_num7 = cfg_get_int(SECTION_START_LOCATION,NULL,"s_num7");

    con->f_loc = cfg_get_int(SECTION_START_LOCATION,NULL,"f_loc");
    con->f_length                                                    =
cfg_get_int(SECTION_START_LOCATION,NULL,"f_length");
    con->f_fixed                                                     =
cfg_get_int(SECTION_START_LOCATION,NULL,"f_fixed");
    con->f_count                                                     =
cfg_get_int(SECTION_START_LOCATION,NULL,"f_count");
    con->f_num0 = cfg_get_int(SECTION_START_LOCATION,NULL,"f_num0");
    con->f_num1 = cfg_get_int(SECTION_START_LOCATION,NULL,"f_num1");
    con->f_num2 = cfg_get_int(SECTION_START_LOCATION,NULL,"f_num2");
    con->f_num3 = cfg_get_int(SECTION_START_LOCATION,NULL,"f_num3");
    con->f_num4 = cfg_get_int(SECTION_START_LOCATION,NULL,"f_num4");
    con->f_num5 = cfg_get_int(SECTION_START_LOCATION,NULL,"f_num5");
    con->f_num6 = cfg_get_int(SECTION_START_LOCATION,NULL,"f_num6");
    con->f_num7 = cfg_get_int(SECTION_START_LOCATION,NULL,"f_num7");

    rec_entry = con->s_loc;
    rec_numbers[0]=con->s_num0;
    rec_numbers[1]=con->s_num1;
    rec_numbers[2]=con->s_num2;
    rec_numbers[3]=con->s_num3;
    rec_numbers[4]=con->s_num4;
    rec_numbers[5]=con->s_num5;
    rec_numbers[6]=con->s_num6;
    rec_numbers[7]=con->s_num7;

      ptr = header->order + (entry = rec_entry) * 3;
      memcpy(numbers, rec_numbers, sizeof(numbers));

      crk_init(db, fix_state, NULL);

      last_count = last_length = -1;

      entry--;
      while (ptr < &header->order[sizeof(header->order) - 1]) {
            con->s_loc = entry++;
            con->s_length = length = *ptr++;
        con->s_fixed = fixed = *ptr++;
        con->s_count = count = *ptr++;

        if (entry == rec_entry){
            if(con->s_length != length)
               log_event("Length does not match");
            if(con->s_fixed != fixed)
                log_event("Fixed does not match");
            if(con->s_count != count)
                log_event("Count does not match");
        }
```

```
            if (length >= CHARSET_LENGTH ||
                  fixed > length ||
                  count >= CHARSET_SIZE) inc_format_error(charset);

            if (entry != rec_entry)
                  memset(numbers, 0, sizeof(numbers));

            if (count >= real_count ||
                  (int)length  >=  db->format->params.plaintext_length
||
                  (fixed && !count)) continue;

            if ((int)length + 1 < min_length ||
                  (int)length >= max_length ||
                  (int)count >= max_count) continue;

            if ((int)length != last_length) {
                  inc_new_length(last_length = length,
                        header, file, charset, char1, char2, chars);
                  last_count = -1;
            }
            if ((int)count > last_count)
                  inc_new_count(length, last_count = count,
                        allchars, char1, char2, chars);

            if (!length && !min_length) {
                  min_length = 1;
                  if (crk_process_key("")) break;
            }

            log_event("- Trying length %d, fixed @%d, character count
%d",
                  length + 1, fixed + 1, count + 1);

            if  (inc_key_loop(length,  fixed,  count,  char1,  char2,
chars,con))
                  break;
            break;
      }

      crk_done();
//    rec_done(event_abort);

      for (pos = 0; pos < (int)header->length - 2; pos++)
            MEM_FREE(chars[pos]);
      MEM_FREE(char2);
      MEM_FREE(header);

      fclose(file);
}
```

H. OPTIONS.H

```
/*
 * This file is part of John the Ripper password cracker,
 * Copyright (c) 1996-98,2003 by Solar Designer
 */

/*
 * John's command line options definition.
 */

#ifndef _JOHN_OPTIONS_H
#define _JOHN_OPTIONS_H

#include "list.h"
#include "loader.h"
#include "getopt.h"

/*
 * Option flags bitmasks.
 */
/* An action requested */
#define FLG_ACTION                0x00000001
/* Password files specified */
#define FLG_PASSWD                0x00000002
/* An option supports password files */
#define FLG_PWD_SUP               0x00000004
/* An option requires password files */
#define FLG_PWD_REQ               (0x00000008 | FLG_PWD_SUP)
/* Some option that doesn't have its own flag is specified */
#define FLG_NONE            0x00000010
/* A cracking mode enabled */
#define FLG_CRACKING_CHK          0x00000020
#define FLG_CRACKING_SUP          0x00000040
#define FLG_CRACKING_SET \
      (FLG_CRACKING_CHK   |   FLG_CRACKING_SUP   |   FLG_ACTION   |
FLG_PWD_REQ)
/* Wordlist mode enabled, options.wordlist is set to the file name
or NULL
 * if reading from stdin. */
#define FLG_WORDLIST_CHK          0x00000080
#define FLG_WORDLIST_SET          (FLG_WORDLIST_CHK               |
FLG_CRACKING_SET)
/* Wordlist mode enabled, reading from stdin */
#define FLG_STDIN_CHK             0x00000100
#define FLG_STDIN_SET             (FLG_STDIN_CHK | FLG_WORDLIST_SET)
/* Wordlist rules enabled */
#define FLG_RULES           0x00000200
/* "Single crack" mode enabled */
#define FLG_SINGLE_CHK            0x00000400
#define FLG_SINGLE_SET            (FLG_SINGLE_CHK                 |
FLG_CRACKING_SET)
/* Incremental mode enabled */
#define FLG_INC_CHK               0x00000800
```

```
#define FLG_INC_SET               (FLG_INC_CHK | FLG_CRACKING_SET)
/* External mode or word filter enabled */
#define FLG_EXTERNAL_CHK          0x00001000
#define FLG_EXTERNAL_SET \
      (FLG_EXTERNAL_CHK    |    FLG_ACTION    |    FLG_CRACKING_SUP    |
FLG_PWD_SUP)
/* Batch cracker */
#define FLG_BATCH_CHK             0x00004000
#define FLG_BATCH_SET             (FLG_BATCH_CHK | FLG_CRACKING_SET)
/* Stdout mode */
#define FLG_STDOUT                0x00008000
/* Restoring an interrupted session */
#define FLG_RESTORE_CHK                0x00010000
#define FLG_RESTORE_SET                (FLG_RESTORE_CHK           |
FLG_ACTION)
/* A session name is set */
#define FLG_SESSION               0x00020000
/* Print status of a session */
#define FLG_STATUS_CHK            0x00040000
#define FLG_STATUS_SET            (FLG_STATUS_CHK | FLG_ACTION)
/* Make a charset */
#define FLG_MAKECHR_CHK                0x00100000
#define FLG_MAKECHR_SET \
      (FLG_MAKECHR_CHK | FLG_ACTION | FLG_PWD_SUP)
/* Show cracked passwords */
#define FLG_SHOW_CHK              0x00200000
#define FLG_SHOW_SET \
      (FLG_SHOW_CHK | FLG_ACTION | FLG_PWD_REQ)
/* Perform a benchmark */
#define FLG_TEST_CHK              0x00400000
#define FLG_TEST_SET \
      (FLG_TEST_CHK | FLG_CRACKING_SUP | FLG_ACTION)
/* Passwords per salt requested */
#define FLG_SALTS           0x01000000
/* Ciphertext format forced */
#define FLG_FORMAT                0x02000000
/* Memory saving enabled */
#define FLG_SAVEMEM               0x04000000

/*
 * Structure with option flags and all the parameters.
 */
struct options_main {
/* Option flags */
      opt_flags flags;

/* Password files */
      struct list_main *passwd;

/* Password file loader options */
      struct db_options loader;

/* Session name */
      char *session;
```

```
/* Ciphertext format name */
      char *format;

/* Wordlist file name */
      char *wordlist;

/* Charset file name */
      char *charset;

/* External mode or word filter name */
      char *external;

/* Maximum plaintext length for stdout mode */
      int length;
};

extern struct options_main options;

/*
 * Initializes the options structure.
 */
extern void opt_init(int argc, char **argv, char *);

#endif
```

I. OPTIONS.C

```
/*
 * This file is part of John the Ripper password cracker,
 * Copyright (c) 1996-2005 by Solar Designer
 */

#include <stdio.h>
#include <stdlib.h>
#include <string.h>

#include "arch.h"
#include "misc.h"
#include "params.h"
#include "memory.h"
#include "list.h"
#include "loader.h"
#include "logger.h"
#include "status.h"
#include "recovery.h"
#include "options.h"
#include "config.h"

struct options_main options;

static struct opt_entry opt_list[] = {
      {"", FLG_PASSWD, 0, 0, 0, OPT_FMT_ADD_LIST, &options.passwd},
```

```
	{"single", FLG_SINGLE_SET, FLG_CRACKING_CHK},
	{"wordlist", FLG_WORDLIST_SET, FLG_CRACKING_CHK,
		0, OPT_REQ_PARAM, OPT_FMT_STR_ALLOC, &options.wordlist},
	{"stdin", FLG_STDIN_SET, FLG_CRACKING_CHK},
	{"rules",	FLG_RULES,	FLG_RULES,	FLG_WORDLIST_CHK,
FLG_STDIN_CHK},
	{"incremental", FLG_INC_SET, FLG_CRACKING_CHK,
		0, 0, OPT_FMT_STR_ALLOC, &options.charset},
	{"external", FLG_EXTERNAL_SET, FLG_EXTERNAL_CHK,
		0, OPT_REQ_PARAM, OPT_FMT_STR_ALLOC, &options.external},
	{"stdout", FLG_STDOUT, FLG_STDOUT,
		FLG_CRACKING_SUP, FLG_SINGLE_CHK | FLG_BATCH_CHK,
		"%u", &options.length},
	{"restore", FLG_RESTORE_SET, FLG_RESTORE_CHK,
		0, ~FLG_RESTORE_SET & ~OPT_REQ_PARAM,
		OPT_FMT_STR_ALLOC, &options.session},
	{"session", FLG_SESSION, FLG_SESSION,
		FLG_CRACKING_SUP, OPT_REQ_PARAM,
		OPT_FMT_STR_ALLOC, &options.session},
	{"status", FLG_STATUS_SET, FLG_STATUS_CHK,
		0, ~FLG_STATUS_SET & ~OPT_REQ_PARAM,
		OPT_FMT_STR_ALLOC, &options.session},
	{"make-charset", FLG_MAKECHR_SET, FLG_MAKECHR_CHK,
		0, FLG_CRACKING_CHK | FLG_SESSION | OPT_REQ_PARAM,
		OPT_FMT_STR_ALLOC, &options.charset},
	{"show", FLG_SHOW_SET, FLG_SHOW_CHK,
		0, FLG_CRACKING_SUP | FLG_MAKECHR_CHK},
	{"test", FLG_TEST_SET, FLG_TEST_CHK,
		0, ~FLG_TEST_SET & ~FLG_FORMAT & ~FLG_SAVEMEM &
		~OPT_REQ_PARAM},
	{"users", FLG_NONE, 0, FLG_PASSWD, OPT_REQ_PARAM,
		OPT_FMT_ADD_LIST_MULTI, &options.loader.users},
	{"groups", FLG_NONE, 0, FLG_PASSWD, OPT_REQ_PARAM,
		OPT_FMT_ADD_LIST_MULTI, &options.loader.groups},
	{"shells", FLG_NONE, 0, FLG_PASSWD, OPT_REQ_PARAM,
		OPT_FMT_ADD_LIST_MULTI, &options.loader.shells},
	{"salts", FLG_SALTS, FLG_SALTS, FLG_PASSWD, OPT_REQ_PARAM,
		"%d", &options.loader.min_pps},
	{"format", FLG_FORMAT, FLG_FORMAT,
		FLG_CRACKING_SUP,
		FLG_MAKECHR_CHK | FLG_STDOUT | OPT_REQ_PARAM,
		OPT_FMT_STR_ALLOC, &options.format},
	{"save-memory", FLG_SAVEMEM, FLG_SAVEMEM, 0, OPT_REQ_PARAM,
		"%u", &mem_saving_level},
	{NULL}
};

#if DES_BS
/* nonstd.c and parts of x86-mmx.S aren't mine */
#define JOHN_COPYRIGHT \
	"Solar Designer and others"
#else
#define JOHN_COPYRIGHT \
	"Solar Designer"
```

```
#endif

#define JOHN_USAGE \
"John the Ripper password cracker, version " JOHN_VERSION "\n" \
"Copyright (c) 1996-2005 by " JOHN_COPYRIGHT "\n" \
"Homepage: http://www.openwall.com/john/\n" \
"\n" \
"Usage: %s [OPTIONS] [PASSWORD-FILES]\n" \
"--single                   \"single crack\" mode\n" \
"--wordlist=FILE --stdin     wordlist mode, read words from FILE or
stdin\n" \
"--rules                     enable word mangling rules for wordlist
mode\n" \
"--incremental[=MODE]           \"incremental\" mode [using section
MODE]\n" \
"--format=NAME              force ciphertext format NAME: " \
      "DES/BSDI/MD5/BF/AFS/LM\n" \
"--save-memory=LEVEL        enable memory saving, at LEVEL 1..3\n"
/*"--external=MODE            external mode or word filter\n" \
"--stdout[=LENGTH]          just output candidate passwords [cut at
LENGTH]\n" \
"--restore[=NAME]            restore an interrupted session [called
NAME]\n" \
"--session=NAME             give a new session the NAME\n" \
"--status[=NAME]                 print status of a session [called
NAME]\n" \
"--make-charset=FILE                make a charset, FILE will be
overwritten\n" \
"--show                     show cracked passwords\n" \
"--test                     perform a benchmark\n" \
"--users=[-]LOGIN|UID[,..]   [do not] load this (these) user(s)
only\n" \
"--groups=[-]GID[,..]           load users [not] of this (these)
group(s) only\n" \
"--shells=[-]SHELL[,..]         load users with[out] this (these)
shell(s) only\n" \
"--salts=[-]COUNT              load salts with[out] at least COUNT
passwords " \
      "only\n" \*/

void opt_init(int argc, char **argv,char *pw_file_name)
{
      int new_argc= 5;
      char *cr_arg;
      char cr_arg_set[254];
      char *cr_arg_rule;
      char cr_arg_rule_set[254];
        char **new_argv;
      int i=0;
      int j;
      fprintf(stderr," Inside opt_init \nargc : %d\n"
                  "argv0: %s \npw_file_name: %s\n",
                  argc, argv[0], pw_file_name);
```

```
      new_argv   =   mem_alloc_tiny(sizeof(char   *)   *   (new_argc),
MEM_ALIGN_WORD);
      new_argv[i++]= str_alloc_copy(argv[0]);
      fprintf(stderr," New_argv0: %s\n",new_argv[0]);

      cr_arg = cfg_get_param(SECTION_CRACK_MODE,NULL,"Mode");
      if (!strncmp(cr_arg,"-wordlist",9)){
            sprintf(cr_arg_set,"%s=%s",cr_arg,WORDLIST_NAME);
            fprintf(stderr,"cr_arg: %s\n",cr_arg_set);
            new_argv[i++]= str_alloc_copy(cr_arg_set);
            fprintf(stderr,"After       str_alloc_copy       cr_arg:
%s\n",new_argv[2]);
            cr_arg_rule                                              =
cfg_get_param(SECTION_CRACK_MODE,NULL,"Req_Mode");
            strcpy(cr_arg_rule_set,cr_arg_rule);
            fprintf(stderr,"cr_arg_rule:%s\n",cr_arg_rule_set);
            new_argv[i++]= str_alloc_copy(cr_arg_rule_set);
      }else{
            new_argv[i++]= str_alloc_copy(cr_arg);
      }

      new_argv[i++]=str_alloc_copy(pw_file_name);
      fprintf(stderr,"before new_argv0 error check\n");
      if (!new_argv[0]) error();

      if (!new_argv[1]) {
            printf(JOHN_USAGE, new_argv[0]);
            exit(0);
      }

      memset(&options, 0, sizeof(options));

      list_init(&options.passwd);

      options.loader.flags = DB_LOGIN;
      fprintf(stderr,"opt.ldr.flg: %d\n",options.loader.flags);
      list_init(&options.loader.users);
      list_init(&options.loader.groups);
      list_init(&options.loader.shells);

      options.length = -1;
      for (j=0;j<5;j++){
            fprintf(stderr,"argv[%d]: %s\n",j,new_argv[j]);
      }
      fprintf(stderr," before opt_process\n");
      opt_process(opt_list, &options.flags, (char**)new_argv);
      fprintf(stderr," After opt_process\n");

      if ((options.flags &
          (FLG_EXTERNAL_CHK  |  FLG_CRACKING_CHK  |  FLG_MAKECHR_CHK))
==
          FLG_EXTERNAL_CHK)
            options.flags |= FLG_CRACKING_SET;
```

```
	if (!(options.flags & FLG_ACTION))
		options.flags |= FLG_BATCH_SET;

	opt_check(opt_list, options.flags, argv);

	if (options.session)
		rec_name = options.session;

	if (options.flags & FLG_RESTORE_CHK) {
		rec_restore_args(1);
		return;
	}

	if (options.flags & FLG_STATUS_CHK) {
		rec_restore_args(0);
		options.flags |= FLG_STATUS_SET;
		status_init(NULL, 1);
		status_print();
		exit(0);
	}

	if (options.flags & FLG_SALTS)
	if (options.loader.min_pps < 0) {
		options.loader.max_pps = -1 - options.loader.min_pps;
		options.loader.min_pps = 0;
	}

	if (options.length < 0)
		options.length = PLAINTEXT_BUFFER_SIZE - 3;
	else
	if (options.length < 1 || options.length >
PLAINTEXT_BUFFER_SIZE - 3) {
		fprintf(stderr, "Invalid plaintext length requested\n");
		error();
	}

	if (options.flags & FLG_STDOUT) options.flags &= ~FLG_PWD_REQ;

	if ((options.flags & (FLG_PASSWD | FLG_PWD_REQ)) ==
FLG_PWD_REQ) {
		fprintf(stderr, "Password files required, "
			"but none specified\n");
		error();
	}

	if ((options.flags & (FLG_PASSWD | FLG_PWD_SUP)) ==
FLG_PASSWD) {
		fprintf(stderr, "Password files specified, "
			"but no option would use them\n");
		error();
	}

	rec_argc = argc; rec_argv = argv;
}
```

J. PARAMS.H

```
/*
 * This file is part of John the Ripper password cracker,
 * Copyright (c) 1996-2005 by Solar Designer
 */

/*
 * Some global parameters.
 */

#ifndef _JOHN_PARAMS_H
#define _JOHN_PARAMS_H

#include <limits.h>

/*
 * John's version number.
 */
#define JOHN_VERSION              "1.6.38"

/*
 *  Is  this  a  system-wide  installation?  *BSD  ports  and  Linux
distributions
 * will probably want to set this to 1 for their builds of John.
 */
#ifndef JOHN_SYSTEMWIDE
#define JOHN_SYSTEMWIDE                 0
#endif

#if JOHN_SYSTEMWIDE
#define JOHN_SYSTEMWIDE_EXEC      "/usr/libexec/john"
#define JOHN_SYSTEMWIDE_HOME      "/usr/share/john"
#define JOHN_PRIVATE_HOME         "~/.john"
#endif

/*
 * Crash recovery file format version strings.
 */
#define RECOVERY_VERSION_0        "REC0"
#define RECOVERY_VERSION_1        "REC1"
#define RECOVERY_VERSION_2        "REC2"
#define RECOVERY_VERSION_CURRENT  RECOVERY_VERSION_2

/*
 * Charset file format version string.
 */
#define CHARSET_VERSION                 "CHR1"

/*
 * Timer interval in seconds.
 */
```

```
#define TIMER_INTERVAL            1

/*
 * Default crash recovery file saving delay in timer intervals.
 */
#define TIMER_SAVE_DELAY          (600 / TIMER_INTERVAL)

/*
 * Benchmark time in seconds, per cracking algorithm.
 */
#define BENCHMARK_TIME            5

/*
 * Number of salts to assume when benchmarking.
 */
#define BENCHMARK_MANY            0x100

/*
 * File names.
 */
#define CFG_FULL_NAME             "john_boinc.conf"
#define CFG_ALT_NAME              "john_boinc.ini"
#if JOHN_SYSTEMWIDE
#define CFG_PRIVATE_FULL_NAME           JOHN_PRIVATE_HOME
"/john.conf"
#define CFG_PRIVATE_ALT_NAME      JOHN_PRIVATE_HOME "/john.ini"
#define POT_NAME             JOHN_PRIVATE_HOME "/john.pot"
#define LOG_NAME             JOHN_PRIVATE_HOME "/john.log"
#define RECOVERY_NAME             JOHN_PRIVATE_HOME "/john.rec"
#else
#define POT_NAME             "$JOHN/john_boinc.pot"
#define LOG_NAME             "$JOHN/john_boinc.log"
#define RECOVERY_NAME             "$JOHN/john_boinc.rec"
#endif
#define LOG_SUFFIX                ."log"
#define RECOVERY_SUFFIX                ."rec"
#define WORDLIST_NAME             "password.lst"
#define OUTPUT_FILENAME                "cr_pw_out"

/*
 * Configuration file section names.
 */
#define SECTION_OPTIONS                "Options"
#define SECTION_RULES             "List.Rules:"
#define SUBSECTION_SINGLE         "Single"
#define SUBSECTION_WORDLIST       "Wordlist"
#define SECTION_INC               "Incremental:"
#define SECTION_EXT               "List.External:"
#define SECTION_CRACK_MODE        "Crack_Mode"
#define SECTION_PASSWD_ID         "List.Password_id"
#define SECTION_START_LOCATION    "Start_Location"
/*
 *  Hash  table  sizes.  These  are  also  hardcoded  into  the  hash
functions.
```

```
 */
#define SALT_HASH_SIZE            0x400
#define PASSWORD_HASH_SIZE_0      0x10
#define PASSWORD_HASH_SIZE_1      0x100
#define PASSWORD_HASH_SIZE_2      0x1000

/*
 * Password hash table thresholds. These are the counts of entries
required
 * to enable the corresponding hash table size.
 */
#define PASSWORD_HASH_THRESHOLD_0 (PASSWORD_HASH_SIZE_0 / 2)
#define PASSWORD_HASH_THRESHOLD_1 (PASSWORD_HASH_SIZE_1 / 4)
#define PASSWORD_HASH_THRESHOLD_2 (PASSWORD_HASH_SIZE_2 / 4)

/*
 * Tables of the above values.
 */
extern int password_hash_sizes[3];
extern int password_hash_thresholds[3];

/*
 * Cracked password hash size, used while loading.
 */
#define CRACKED_HASH_LOG          10
#define CRACKED_HASH_SIZE         (1 << CRACKED_HASH_LOG)

/*
 * Password hash function to use while loading.
 */
#define LDR_HASH_SIZE  (PASSWORD_HASH_SIZE_2     *     sizeof(struct
db_password *))
#define LDR_HASH_FUNC  (format->methods.binary_hash[2])

/*
 * Buffered keys hash size, used for "single crack" mode.
 */
#define SINGLE_HASH_LOG               5
#define SINGLE_HASH_SIZE          (1 << SINGLE_HASH_LOG)

/*
 * Minimum buffered keys hash size, used if min_keys_per_crypt is
even less.
 */
#define SINGLE_HASH_MIN               8

/*
 * Shadow file entry table hash size, used by unshadow.
 */
#define SHADOW_HASH_LOG               8
#define SHADOW_HASH_SIZE          (1 << SHADOW_HASH_LOG)

/*
 * Hash and buffer sizes for unique.
```

```
 */
#define UNIQUE_HASH_LOG                17
#define UNIQUE_HASH_SIZE           (1 << UNIQUE_HASH_LOG)
#define UNIQUE_BUFFER_SIZE         0x800000

/*
 * Maximum number of GECOS words per password to load.
 */
#define LDR_WORDS_MAX              0x10

/*
 * Maximum number of GECOS words to try in pairs.
 */
#define SINGLE_WORDS_PAIR_MAX           4

/*
 * Charset parameters.
 * Be careful if you change these, ((SIZE ** LENGTH) * SCALE) should
fit
 * into 64 bits. You can reduce the SCALE if required.
 */
#define CHARSET_MIN                ' '
#define CHARSET_MAX                0x7E
#define CHARSET_SIZE               (CHARSET_MAX - CHARSET_MIN + 1)
#define CHARSET_LENGTH             8
#define CHARSET_SCALE              0x100

/*
 * Compiler parameters.
 */
#define C_TOKEN_SIZE               0x100
#define C_UNGET_SIZE               (C_TOKEN_SIZE + 4)
#define C_EXPR_SIZE                0x100
#define C_STACK_SIZE               ((C_EXPR_SIZE + 4) * 4)
#define C_ARRAY_SIZE               0x1000000
#define C_DATA_SIZE                0x8000000

/*
 * Buffer size for rules.
 */
#define RULE_BUFFER_SIZE           0x100

/*
 * Maximum number of character ranges for rules.
 */
#define RULE_RANGES_MAX                 8

/*
 * Buffer size for words while applying rules, should be at least as
large
 * as PLAINTEXT_BUFFER_SIZE.
 */
#define RULE_WORD_SIZE             0x80
```

```
/*
 * Buffer size for plaintext passwords.
 */
#define PLAINTEXT_BUFFER_SIZE           0x80

/*
 * Buffer size for fgets().
 */
#define LINE_BUFFER_SIZE          0x400

/*
 * john.pot and log file buffer sizes, can be zero.
 */
#define POT_BUFFER_SIZE                 0x1000
#define LOG_BUFFER_SIZE                 0x1000

/*
 * Buffer size for path names.
 */
#ifdef PATH_MAX
#define PATH_BUFFER_SIZE          PATH_MAX
#else
#define PATH_BUFFER_SIZE          0x400
#endif

#endif
```

K. SINGLE.H

```
/*
 * This file is part of John the Ripper password cracker,
 * Copyright (c) 1996-98 by Solar Designer
 */

/*
 * "Single crack" mode.
 */

#ifndef _JOHN_SINGLE_H
#define _JOHN_SINGLE_H

#include "loader.h"

/*
 * Runs the cracker.
 */
extern void do_single_crack(struct db_main *db);

#endif
```

L. SINGLE.C

```
/*
 * This file is part of John the Ripper password cracker,
 * Copyright (c) 1996-99,2003,2004 by Solar Designer
 */

#include <stdio.h>
#include <string.h>

#include "misc.h"
#include "params.h"
#include "memory.h"
#include "signals.h"
#include "loader.h"
#include "logger.h"
#include "status.h"
#include "recovery.h"
#include "rpp.h"
#include "rules.h"
#include "external.h"
#include "cracker.h"

static int progress = 0;
static int rec_rule;

static struct db_main *single_db;
static int rule_number, rule_count;
static int length, key_count;
static struct db_keys *guessed_keys;
static struct rpp_context *rule_ctx;

static void save_state(FILE *file)
{
	fprintf(file, "%d\n", rec_rule);
}

static int restore_rule_number(void)
{
	if (rule_ctx)
	for (rule_number = 0; rule_number < rec_rule; rule_number++)
	if (!rpp_next(rule_ctx)) return 1;

	return 0;
}

static int restore_state(FILE *file)
{
	if (fscanf(file, "%d\n", &rec_rule) != 1) return 1;

	return restore_rule_number();
}

static int get_progress(void)
```

```
{
	if (progress) return progress;

	return rule_number * 100 / (rule_count + 1);
}

static void single_alloc_keys(struct db_keys **keys)
{
	int hash_size = sizeof(struct db_keys_hash) +
		sizeof(struct db_keys_hash_entry) * (key_count - 1);

	if (!*keys) {
		*keys = mem_alloc_tiny(
			sizeof(struct db_keys) - 1 + length * key_count,
			MEM_ALIGN_WORD);
		(*keys)->hash = mem_alloc_tiny(hash_size,
MEM_ALIGN_WORD);
	}

	(*keys)->count = 0;
	(*keys)->ptr = (*keys)->buffer;
	(*keys)->rule = rule_number;
	(*keys)->lock = 0;
	memset((*keys)->hash, -1, hash_size);
}

static void single_init(void)
{
	struct db_salt *salt;

	log_event("Proceeding with \"single crack\" mode");

	progress = 0;

	length = single_db->format->params.plaintext_length;
	key_count = single_db->format->params.min_keys_per_crypt;
	if (key_count < SINGLE_HASH_MIN) key_count = SINGLE_HASH_MIN;

	if (rpp_init(rule_ctx, SUBSECTION_SINGLE)) {
		log_event("! No \"single crack\" mode rules found");
		fprintf(stderr, "No \"single crack\" mode rules found in
%s\n",
			cfg_name);
		error();
	}

	rules_init(length);
	rec_rule = rule_number = 0;
	rule_count = rules_count(rule_ctx, 0);

	log_event("- %d preprocessed word mangling rules",
rule_count);

	status_init(get_progress, 0);
```

```
	rec_restore_mode(restore_state);
	rec_init(single_db, save_state);

	salt = single_db->salts;
	do {
		single_alloc_keys(&salt->keys);
	} while ((salt = salt->next));

	if (key_count > 1)
	log_event("-   Allocated   %d   buffer%s   of   %d   candidate
passwords%s",
		single_db->salt_count,
		single_db->salt_count != 1 ? "s" : "",
		key_count,
		single_db->salt_count != 1 ? " each" : "");

	guessed_keys = NULL;
	single_alloc_keys(&guessed_keys);

	crk_init(single_db, NULL, guessed_keys);
}

static int single_key_hash(char *key)
{
	int pos, hash = 0;

	for (pos = 0; pos < length && *key; pos++) {
		hash <<= 1;
		hash ^= *key++;
	}

	hash ^= hash >> SINGLE_HASH_LOG;
	hash ^= hash >> (2 * SINGLE_HASH_LOG);
	hash &= SINGLE_HASH_SIZE - 1;

	return hash;
}

static int single_add_key(struct db_keys *keys, char *key)
{
	int index, hash;
	struct db_keys_hash_entry *entry;

	if ((index = keys->hash->hash[single_key_hash(key)]) >= 0)
	do {
		entry = &keys->hash->list[index];
		if (!strncmp(key, &keys->buffer[entry->offset], length))
			return 0;
	} while ((index = entry->next) >= 0);

	index = keys->hash->hash[hash = single_key_hash(keys->ptr)];
	if (index == keys->count)
		keys->hash->hash[hash] = keys->hash->list[index].next;
```

```
	else
	if (index >= 0) {
		entry = &keys->hash->list[index];
		while ((index = entry->next) >= 0) {
			if (index == keys->count) {
				entry->next = keys->hash->list[index].next;
				break;
			}
			entry = &keys->hash->list[index];
		}
	}

	index = keys->hash->hash[hash = single_key_hash(key)];
	entry = &keys->hash->list[keys->count];
	entry->next = index;
	entry->offset = keys->ptr - keys->buffer;
	keys->hash->hash[hash] = keys->count;

	strnfcpy(keys->ptr, key, length);
	keys->ptr += length;

	return ++(keys->count) >= key_count;
}

static int single_process_buffer(struct db_salt *salt)
{
	struct db_salt *current;
	struct db_keys *keys;
	size_t size;

	if (crk_process_salt(salt)) return 1;

	keys = salt->keys;
	keys->count = 0;
	keys->ptr = keys->buffer;
	keys->lock++;

	if (guessed_keys->count) {
		keys = mem_alloc(size = sizeof(struct db_keys) - 1 +
			length * guessed_keys->count);
		memcpy(keys, guessed_keys, size);

		keys->ptr = keys->buffer;
		do {
			current = single_db->salts;
			do {
				if (current == salt) continue;
				if (!current->list) continue;

				if (single_add_key(current->keys, keys->ptr))
				if (single_process_buffer(current)) return 1;
			} while ((current = current->next));
			keys->ptr += length;
		} while (--keys->count);
```

```
            MEM_FREE(keys);
      }

      keys = salt->keys;
      keys->lock--;
      if (!keys->count && !keys->lock) keys->rule = rule_number;

      return 0;
}

static   int   single_process_pw(struct   db_salt   *salt,   struct
db_password *pw,
      char *rule)
{
      struct db_keys *keys;
      struct list_entry *first, *second;
      int first_number, second_number;
      char pair[RULE_WORD_SIZE];
      int split;
      char *key;
      unsigned int c;

      keys = salt->keys;

      first_number = 0;
      if ((first = pw->words->head))
      do {
            if ((key = rules_apply(first->data, rule, 0)))
            if (ext_filter(key))
            if (single_add_key(keys, key))
            if (single_process_buffer(salt)) return 1;
            if (!salt->list) return 0;

            if (++first_number > SINGLE_WORDS_PAIR_MAX) continue;

            c = (unsigned int)first->data[0] | 0x20;
            if (c < 'a' || c > 'z') continue;

            second_number = 0;
            second = pw->words->head;

            do
            if (first != second) {
                  if ((split = strlen(first->data)) < length) {
                        strnzcpy(pair, first->data, RULE_WORD_SIZE);
                        strnzcat(pair, second->data, RULE_WORD_SIZE);

                        if ((key = rules_apply(pair, rule, split)))
                        if (ext_filter(key))
                        if (single_add_key(keys, key))
                        if (single_process_buffer(salt)) return 1;
                        if (!salt->list) return 0;
                  }
```

```
                if (first->data[1]) {
                        pair[0] = first->data[0];
                        pair[1] = 0;
                        strnzcat(pair, second->data, RULE_WORD_SIZE);

                        if ((key = rules_apply(pair, rule, 1)))
                        if (ext_filter(key))
                        if (single_add_key(keys, key))
                        if (single_process_buffer(salt)) return 1;
                        if (!salt->list) return 0;
                }
        } while (++second_number <= SINGLE_WORDS_PAIR_MAX &&
                (second = second->next));
    } while ((first = first->next));

    return 0;
}

static int single_process_salt(struct db_salt *salt, char *rule)
{
    struct db_keys *keys;
    struct db_password *pw;

    keys = salt->keys;

    pw = salt->list;
    do {
        if (single_process_pw(salt, pw, rule)) return 1;
        if (!salt->list) return 0;
    } while ((pw = pw->next));

    if (keys->count && rule_number - keys->rule > (key_count <<
1))
        if (single_process_buffer(salt)) return 1;

    if (!keys->count) keys->rule = rule_number;

    return 0;
}

static void single_run(void)
{
    char *prerule, *rule;
    struct db_salt *salt;
    int min, saved_min;

    saved_min = rec_rule;
    while ((prerule = rpp_next(rule_ctx))) {
        if (!(rule = rules_reject(prerule, single_db))) {
            log_event("- Rule #%d: '%.100s' rejected",
                ++rule_number, prerule);
            continue;
        }
```

```
			if (strcmp(prerule, rule))
				log_event("- Rule #%d: '%.100s' accepted as '%s'",
					rule_number + 1, prerule, rule);
			else
				log_event("- Rule #%d: '%.100s' accepted",
					rule_number + 1, prerule);

			if (saved_min != rec_rule) {
				log_event("- Oldest still in use is now rule #%d",
					rec_rule + 1);
				saved_min = rec_rule;
			}

			min = rule_number;

			salt = single_db->salts;
			do {
				if (!salt->list) continue;
				if (single_process_salt(salt, rule)) return;
				if (salt->keys->rule < min) min = salt->keys->rule;
			} while ((salt = salt->next));

			rec_rule = min;
			rule_number++;
		}
	}

	static void single_done(void)
	{
		struct db_salt *salt;

		if (!event_abort) {
			log_event("- Processing the remaining buffered "
				"candidate passwords");

			if ((salt = single_db->salts))
			do {
				if (!salt->list) continue;
				if (salt->keys->count)
				if (single_process_buffer(salt)) break;
			} while ((salt = salt->next));

			progress = 100;
		}

		rec_done(event_abort);
	}

	void do_single_crack(struct db_main *db)
	{
		struct rpp_context ctx;

		single_db = db;
```

```
	rule_ctx = &ctx;
	fprintf(stderr,"before single_init\n");
	single_init();
	fprintf(stderr,"before single_run\n");
	single_run();
	fprintf(stderr,"before single_done\n");
	single_done();
}
```

M. WORDLIST.H

```
/*
 * This file is part of John the Ripper password cracker,
 * Copyright (c) 1996-98 by Solar Designer
 */

/*
 * Wordlist cracker.
 */

#ifndef _JOHN_WORDLIST_H
#define _JOHN_WORDLIST_H

#include "loader.h"

/*
 * Runs the wordlist cracker reading words from the supplied file
name, or
 * stdin if name is NULL.
 */
extern void do_wordlist_crack(struct db_main *db, char *name, int
rules);

#endif
```

N. WORDLIST.C

```
/*
 * This file is part of John the Ripper password cracker,
 * Copyright (c) 1996-99,2003,2004 by Solar Designer
 */

#include <stdio.h>
#include <sys/stat.h>
#include <unistd.h>
#include <string.h>
#include <filesys.h>
#include <boinc_api.h>

#include "misc.h"
#include "math.h"
#include "params.h"
```

```
#include "path.h"
#include "signals.h"
#include "loader.h"
#include "logger.h"
#include "status.h"
#include "recovery.h"
#include "rpp.h"
#include "rules.h"
#include "external.h"
#include "cracker.h"

static FILE *word_file = NULL;
static int progress = 0;

static int rec_rule;
static long rec_pos;

static int rule_number, rule_count, line_number;
static int length;
static struct rpp_context *rule_ctx;

static void save_state(FILE *file)
{
	fprintf(file, "%d\n%ld\n", rec_rule, rec_pos);
}

static int restore_rule_number(void)
{
	if (rule_ctx)
	for (rule_number = 0; rule_number < rec_rule; rule_number++)
	if (!rpp_next(rule_ctx)) return 1;

	return 0;
}

static void restore_line_number(void)
{
	char line[LINE_BUFFER_SIZE];

	for (line_number = 0; line_number < rec_pos; line_number++)
	if (!fgets(line, sizeof(line), word_file)) {
		if (ferror(word_file))
			pexit("fgets");
		else {
			fprintf(stderr, "fgets: Unexpected EOF\n");
			error();
		}
	}
}

static int restore_state(FILE *file)
{
	if (fscanf(file, "%d\n%ld\n", &rec_rule, &rec_pos) != 2)
return 1;
```

```
	if (restore_rule_number()) return 1;

	if (word_file == stdin)
		restore_line_number();
	else
		if (fseek(word_file, rec_pos, SEEK_SET)) pexit("fseek");

	return 0;
}

static void fix_state(void)
{
	rec_rule = rule_number;

	if (word_file == stdin)
		rec_pos = line_number;
	else
	if ((rec_pos = ftell(word_file)) < 0) {
#ifdef __DJGPP__
		if (rec_pos != -1)
			rec_pos = 0;
		else
#endif
			pexit("ftell");
	}
}

static int get_progress(void)
{
	struct stat file_stat;
	long pos;
	int64 x100;

	if (!word_file) return progress;

	if (word_file == stdin) return -1;

	if (fstat(fileno(word_file), &file_stat)) pexit("fstat");

	if ((pos = ftell(word_file)) < 0) {
#ifdef __DJGPP__
		if (pos != -1)
			pos = 0;
		else
#endif
			pexit("ftell");
	}

	mul32by32(&x100, pos, 100);
	return
		(rule_number * 100 +
		div64by32lo(&x100, file_stat.st_size + 1)) / rule_count;
}
```

```
static char *dummy_rules_apply(char *word, char *rule, int split)
{
      word[length] = 0;

      return word;
}

void do_wordlist_crack(struct db_main *db, char *name, int rules)
{
      char line[LINE_BUFFER_SIZE];
      struct rpp_context ctx;
      char *prerule, *rule, *word;
      char last[RULE_WORD_SIZE];
      char *(*apply)(char *word, char *rule, int split);
      int client_ruleid=0;
      char res_name[256];

      log_event("Proceeding with wordlist mode");

      if (boinc_resolve_filename(name,res_name,sizeof(res_name)))
            fprintf(stderr,"Failed to resolve  %s\n",name);

      if (name) {
            if (!(word_file = boinc_fopen(name, "r")))
                  pexit("fopen: %s", name);
            log_event("- Wordlist file: %.100s", name);
      } else {
            word_file = stdin;
            log_event("- Reading candidate passwords from stdin");
      }

      length = db->format->params.plaintext_length;

      if (rules) {
            if (rpp_init(rule_ctx = &ctx, SUBSECTION_WORDLIST)) {
                  log_event("! No wordlist mode rules found");
                  fprintf(stderr, "No wordlist mode rules found in
%s\n",
                        cfg_name);
                  error();
            }

            rules_init(length);
            rule_count = 1;//rules_count(&ctx, -1);

            log_event("- %d preprocessed word mangling rules",
rule_count);

            apply = rules_apply;
      } else {
            rule_ctx = NULL;
            rule_count = 1;
```

```
            log_event("- No word mangling rules");

            apply = dummy_rules_apply;
      }

      client_ruleid=
cfg_get_int(SECTION_START_LOCATION,NULL,"s_loc");
      line_number = rule_number = 0;
      fprintf(stderr,"Looking for ruleid: %d\n",client_ruleid);
      if (rule_ctx)
      for (;rule_number<client_ruleid;rule_number++)
            prerule=rpp_next(rule_ctx);
      else{
            prerule="";
            fprintf(stderr,"NO rule_ctx defined\n");
      }

      status_init(get_progress, 0);

//    rec_restore_mode(restore_state);
 //   rec_init(db, save_state);

      crk_init(db, fix_state, NULL);

/*    if (rules) prerule = rpp_next(&ctx); else prerule = "";
*/    rule = "";

      memset(last, ' ', length + 1);
      last[length + 2] = 0;

      if (prerule)
//    do {
            if (rules) {
                  if ((rule = rules_reject(prerule, db))) {
                        if (strcmp(prerule, rule))
                              fprintf(stderr,"- Rule #%d: '%.100s'"
                                    " accepted as '%.100s'\n",
                                    rule_number , prerule, rule);
                        else
                              fprintf(stderr,"- Rule #%d: '%.100s'"
                                    " accepted\n",
                                    rule_number , prerule);
                  } else
                        fprintf(stderr,"-    Rule    #%d:    '%.100s'
rejected\n",
                              rule_number , prerule);
            }

            if (rule)
            while (fgetl(line, sizeof(line), word_file)) {
                  line_number++;

                  if (line[0] == '#')
                  if (!strncmp(line, "#!comment", 9)) continue;
```

```
                if ((word = apply(line, rule, -1)))
                if (strcmp(word, last)) {
                        strcpy(last, word);

                        if (ext_filter(word))
                        if (crk_process_key(word)) {
                                rules = 0;
                                break;
                        }
                }
        }

/*      if (rules) {
                if (!(rule = rpp_next(&ctx))) break;
                rule_number++;

                line_number = 0;
                if (fseek(word_file, 0, SEEK_SET)) pexit("fseek");
        }
    } while (rules);*/

    crk_done();
//  rec_done(event_abort);

    if (ferror(word_file)) pexit("fgets");

    if (name) {
        if (event_abort)
                progress = get_progress();
        else
                progress = 100;

        if (fclose(word_file)) pexit("fclose");
        word_file = NULL;
    }
}
```

THIS PAGE INTENTIONALLY LEFT BLANK

INITIAL DISTRIBUTION LIST

1. Defense Technical Information Center
   Ft. Belvoir, Virginia

2. Dudley Knox Library
   Naval Postgraduate School
   Monterey, California

3. Dr. George Dinolt
   Naval Postgraduate School
   Monterey, California

4. Dr. Chris Eagle
   Naval Postgraduate School
   Monterey, California